HOBBY COMPUTERS ARE HERE!

$4.95

Edited by Wayne Green W2NSD/1

A
73
PUBLICATION

HOBBY COMPUTERS ARE HERE!

The day of the hobby computer has arrived. This book contains simplified introductions to various aspects of hobby computing, complete with home building projects to get you started. This book is one of the few (if there are any other) sources of information on all aspects of computers . . . hardware (with very simple explanations of the basic circuits involved) . . . the software . . . *and* systems.

Editor: Wayne Green W2NSD/1

73 Magazine – Peterborough NH 03458
First Reprint – 1977
Printed in USA

HOBBY COMPUTERS ARE HERE!

Preface

Way back in 1975, for those of you with long memories, there were hundreds of computer hobbyists. In 1976 there were thousands, and by the end of 1977 there will probably be tens of thousands. But even this is insignificant when the market for the small computers which hobbyists are fanning really develops.

Small, powerful, yet inexpensive computers are being developed for the hobby market and these will inevitably open up the small business, school and home markets for computers. Thus a market that will perhaps amount to $20 million in 1976 may grow to as much as $10 billion by 1980. Growth like that means there is a lot of money in there for anyone willing to take a crack at it . . . but before you can shake the golden goose you have to get a halter on it . . . and the fact is that simple to understand books about small computer systems are most difficult to find.

Since most of the applications for hobby computers other than the playing of games have been developed by amateur radio operators, it is not surprising that much of the beginning literature of the field has appeared in an amateur radio magazine: 73 Magazine. Indeed, interest in hobby computing has been so high among amateurs that 73 has been devoting a whole section of the magazine to the subject.

The serious entrepreneur of next year will be the person who has a good understanding of computers . . . hardware, software, *and* systems. Oddly enough, about the *only* way to get such knowledge today is via hobby computing. Computer professionals working for the larger companies have little opportunity to get expertise and true understanding outside of their narrow field of work. The hobbyist of today has by far the best opportunity to be the millionaire of next year . . . or at least the year after, if he is a slow worker.

HOBBY COMPUTERS ARE HERE!

Wayne Green's Editorials

Wayne Green W2NSD/1
c/o 73 Magazine
Peterborough NH 03458

Computers Are Here - Are You Ready?

Like Duz (do they still make Duz?), computers are getting a reputation for being able to do just about everything. It is well earned, for to understand computers is to love them . . . they are being billed as the World's Greatest Toy, and this is not much an exaggeration, young long-haired blonds notwithstanding.

More and more amateurs are tackling the new inexpensive computer kits and coming up with very usable results. Some are using the units to aim their antennas for moonbounce, some to predict or even aim antennas at Oscar, some to operate a virtually automatic RTTY station, some to run a repeater or even a system of repeaters . . . and so forth.

The Three Basics

There are three parts to a computer system . . . the central processing unit, cleverly called a CPU, the gadget which costs the most money and which does most of the work . . . an input/output device such as a teletype . . . and some sort of memory for the CPU to keep things on file when it is not actively working with them.

IC technology has been raising havoc with CPU prices, dropping them in large increments every few months. The latest chips such as the Intel 8008, 8080, National PACE, IMP-16, and Motorola M6800 have spawned a breed of miniature CPU which is so low in cost that it has made the hobby computer a practicality. The first large quantity production of CPUs using the new series of microprocessor chips was put out by MITS in January . . . their Altair 8800. This sold for $439 in kit form and $621 assembled and tested . . . about one tenth the price of previously available minicomputer units.

RGS Electronics and Scelbi Computer Systems had been producing computer kits before this using the Intel 8008 chip, but these were not as well publicized and the 8008 chip has more limitations than the 8080, which is used in the Altair.

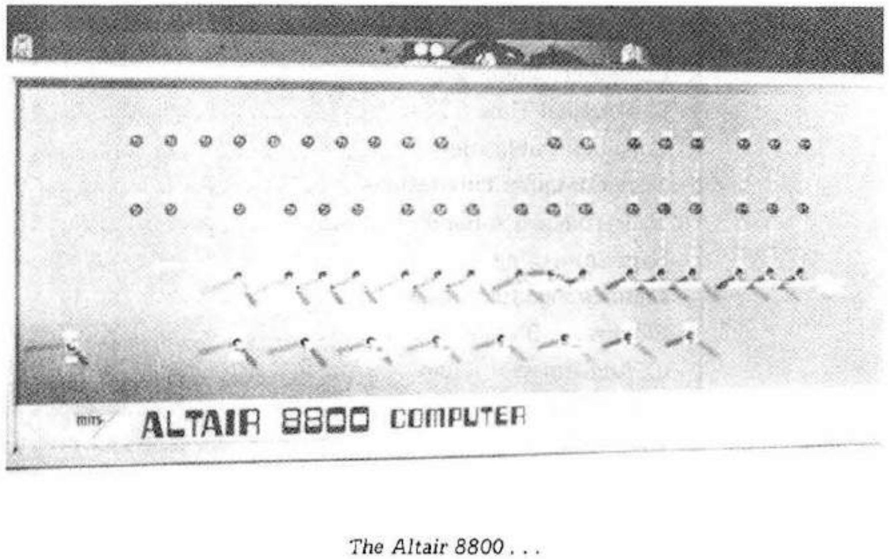
More and more CPU kits are becoming available . . . such as the recently announced Godbout system using the National PACE chip . . . this holds a lot of promise for a lot of computer at a ridiculously low price. Another new one is Sphere, available in kit or assembled form in the same basic price range of $500-$600 with enough built-in memory to do some work.

There are two basic forms of memory required . . . one in the CPU to permit it to do its work . . . and one outside for longer term use. The internal memory stores operating program instructions and things retrieved from the larger memory which have to be used by the CPU. Practically speaking, the larger the internal memory of the CPU, the faster your computer system can operate. For instance, if you had a record of all of the stations you've ever contacted in the main memory and you wanted to sort through for one particular call, it would be easier to find if your CPU could grab a thousand stations out at one time and check them against the call you need instead of checking maybe ten at one time. You'd find the record you want one hundred times faster.

But, alas, memory costs money, and a happy medium has to be struck between what you want and what you can afford. You can get along with 4k of memory . . . that's actually 4096 bytes, where a byte is 8 bits of memory, the amount needed to represent a letter or number. Memory costs about 4¢ per byte today, but it will be coming down.

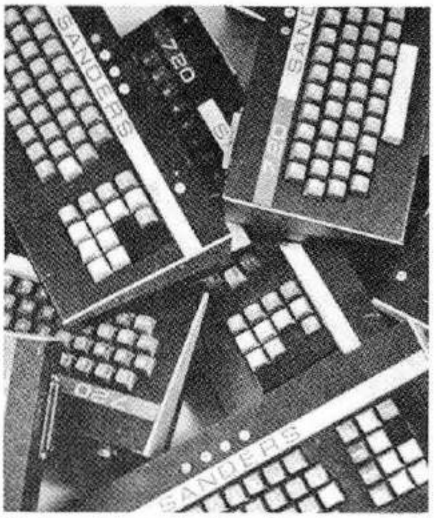
Long term memory units have been coming down in price too, though it is still possible to buy a brand new Ampex 40 Megabyte disk system for $24,000 if you like to pay list price and get into that sort of scene. More in the amateur end are some of the soon to be seen floppy disk systems which will be selling in the $500 range and which will provide about 250k of memory on each disk. The disks are a lot like phonograph records and can be changed quickly.

One of the simplest and least expensive memory systems involves the audio cassette recorder, and many hobbyists seem to be



The Altair 8800 . . .

working in this direction. It is a little slow, but it is extremely cheap and you can have a lot of memory that way. The standard for using them is a familiar one, with the two RTTY audio frequency shift tones being used.



Keyboards from Sanders: Running rampant.

Even the input/output situation is changing rapidly. Of course you can buy an old teletype machine for $50 to $100 and it will work quite well. You might even want to go to a faster and more modern machine, if you can promote one at less than the price of a good used car. The more usual system now is to put together a video display terminal and work from that. These are available in kit form for around $150 to $250 and are a cinch to put together. The Southwest Technical video display generator costs $175 in kit form and can put together by a 12 year old. The keyboard that goes with this one runs another $40 and works like a champ. Or you may want to shop around for a surplus keyboard for the same or a slightly lower price ... most of them have ASCII output and this is all you need to hook things together for a working system. (American Standard Code for Information Interchange = ASCII.)

Uses

Once you have your CPU, memory and I/O up and working, you then have to decide what you want to do with the system. You may want to use it to keep track of stations you've worked, with little bits of information about them for recall on the video screen (any television set will provide the video part of the terminal for you). You may want to catalog your record collection or your book library ... or perhaps articles in the ham magazines.

If you are into RTTY you realize that your computer system is the main part of a RTTY station. You can program it to send at 60 words per minute, either from the keyboard or from any material you have in the memory ... and receive the same way, printing it out (and memorizing the stuff, if you want) on your screen.

Perhaps you prefer CW ... so program the computer to convert the ASCII letters into appropriate CW characters ... select the speed you prefer ... and type away as fast as you like for several hundred words. Your computer can also decipher incoming CW for you and print it on the screen. There probably will be a good deal of 50 wpm CW around in the future as computer-assisted ops work each other.

Your checkbook? No strain, many hobbyists are using their systems for keeping their bank accounts in order.

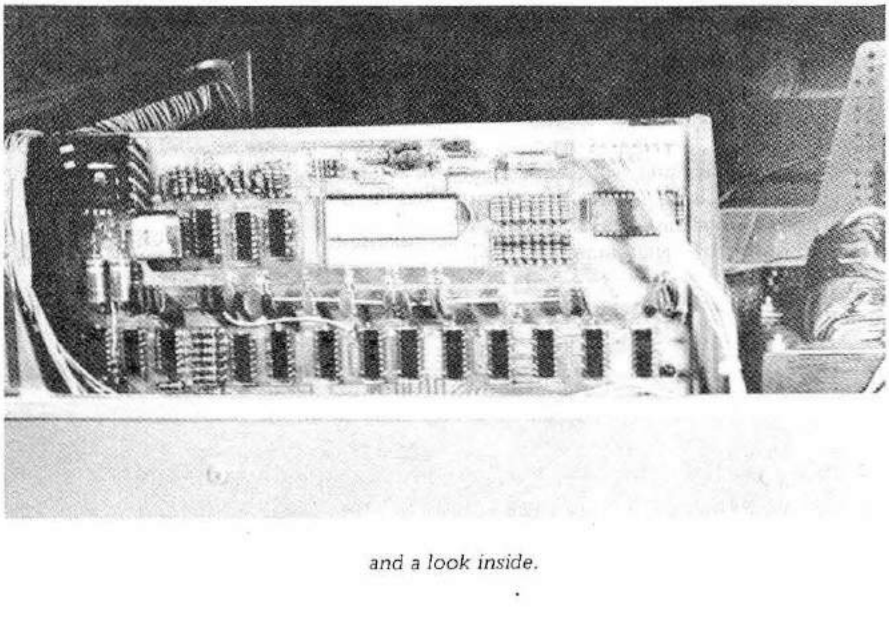
If you have a small business of your own you may want to apply some of the computer power to it ... inventory ... accounts receivable ... mailing lists ... things like that.

Programming

This is a bit sticky right now, but the situation is improving. One problem has been that there have been a whole lot of computers designed, but not all that many of any one model ... so the programming work has had to be done over and over to match each new machine that has been developed. And as computers have gotten more and more complicated, programming has followed ... usually increasing about ten times in difficulty for each increase in complexity of the CPU. Thus, while CPU costs have been dropping rapidly, programming (software) costs have been going up by like amounts. This may improve as more and more identical computers are made available ... a benefit of mass production.

The problem with programming is that it takes forever to put in instructions when you have to do it one single step at a time. The idea is to enable the computer to translate simple words and instructions into all of the ones and zeroes which the machine requires to do its digital job. These simple yesses and nos are called machine language ... it is the only language the CPU will understand until you "teach" it (via a program) a more complicated set of instructions, thus enabling it to translate.

The manufacturers of most computers spend a lot of time and money working out the translations (programs) and they are usually reluctant to give these away. MITS has implemented Basic and has it available for about $60 when you buy a 4k memory board. A number of small groups have formed to provide cooperative efforts on developing programming and many of the more popular computer systems have user groups who swap programs.



and a look inside.

Learning About Computers

An unfortunate number of the books which have been published with the purported aim of helping you to learn about computers are just plain terrible. The fact is that the rank newcomer to computers is in for a very difficult time. The magazines (with the exception of BYTE, which sort of resulted from this situation) are written for professionals and have little of interest or value to a beginner. Little is written for the experimenter, the circuit designer, or the programmer ... with most magazines being devoted to the business end of the computer field.

73 author Pete Stark has written an interesting introduction to computer programming which is scheduled to be reprinted by Tab Books ... watch for an announcement of that one. The Lancaster TTL Cookbook is fine for hardware fans ... published by Sams at $8.95.

Adding Two Plus Two

The basic CPU usually comes with a set of switches, one for each of the eight bits which make up each byte of information (each character). One set of LED lights indicates which memory location is open for use and the other set indicates what is in that memory at the time. To machine program such a CPU you flip the "examine" switch and the lights will then indicate what is in the first memory position. You set the switches according to the instructions that tell the CPU what you want it to do. Let's say you want to do something very simple at first like add two numbers. Here's how you'd go about it.

There are eight bits to be put in each memory bin. To save a lot of writing and work these are abbreviated. Written out they would look like this:

00 000 000

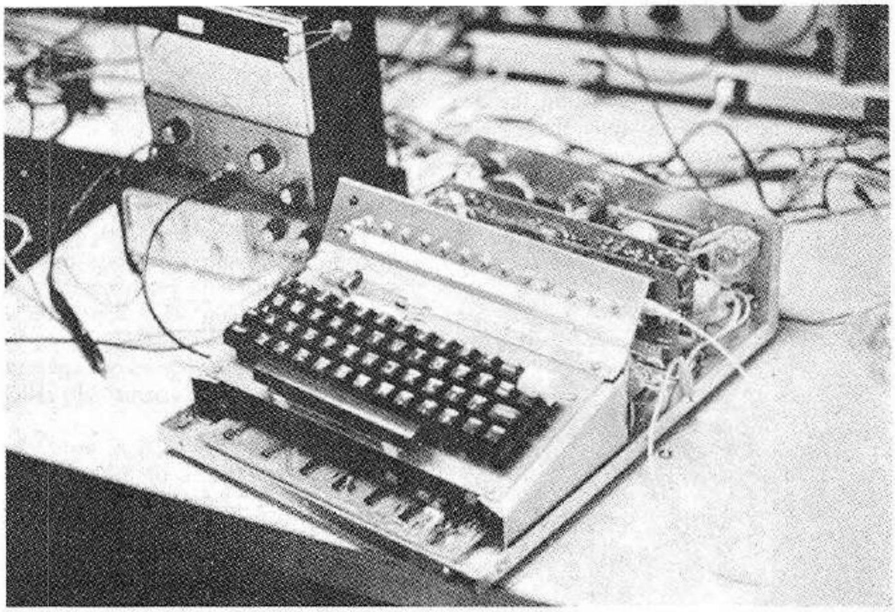
If you've read about binary numbers you know that 000 = 0, 001 = 1, 010 = 2, 011 = 3, 100 = 4, 101 = 5, 110 = 6, 111 = 7. Thus, using this notation, a binary number such as 01 011 111 would be written 137. Got it? That's called octal notation (base 8).

The Intel 8080 chip in the MITS Altair 8800 has a bunch of instructions built into it when you get the unit. This tells you that, if you set the first memory to 072, this will instruct the CPU to pick up whatever number you have in a specific part of the memory and put it into a small working memory unit called register A . . . this is all done inside the 8080 chip. When we push the "deposit" switch this puts the 072 into the first memory position. Next we set up the switches for 200 . . . this is the place where we will put the number we want to add. We push the "deposit next" switch and this puts the 200 into the second memory position. After consulting the instructions again we set up 107, which means move what we had in register A to register B. "Deposit next" takes care of that.

Now we pick up the second number . . . the one we want to add to the first. 072 (deposit next) says move to register A the contents of what we say next . . . 201 (deposit next). A 200 instruction tells the chip to add A to B. Then an 062 tells the chip to move that sum to whatever location we say next . . . we pick 202 and deposit that information. We end the operation by telling the chip to jump back to 000 again and wait for further instructions.

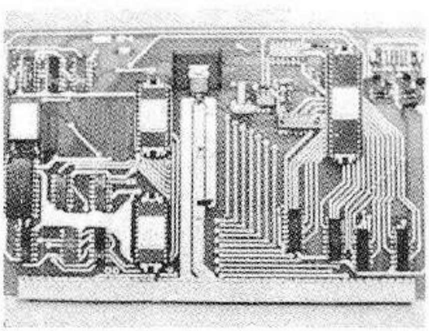
All you have to do to add is put number 1 into location 200, number 2 into location 201 . . . push the "run" and then the "stop" switches and turn the switches to read out what is in memory position 202 . . . presto: the sum of the two numbers.

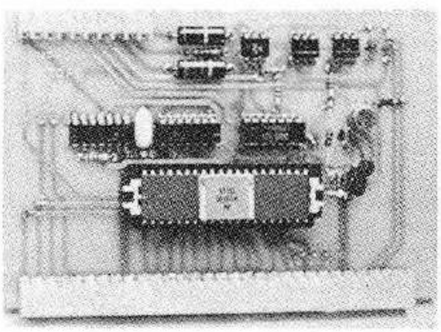
Perhaps you can see why machine language is a back breaker and why everyone wants to get programs which will enable them to merely type in something simple like: A = 300, B = 5.3, C = A x B, PRINT C, END, RUN. There are a great many computer languages . . . something over 700 in current use . . . but a few have come into more popular acceptance such as RPG, Cobol, Basic, Fortran, and such.



A typical test set-up – thorough – at SWT.

The most basic of instructions to the computer are usually put in by means of punched tape or by cassette . . . such simple programs as assemblers. This is a lot faster than sitting there flicking bat switches for each of the eight bits to load a program into a couple thousand memory positions . . . your fingers and patience would wear out. So would the switches after a while.





CPU and Serial Interface boards from Southwest Tech.

If you get into this field you'll find that most of the computer languages are relatively simple . . . you just have to sit down for a few days and work with them until you get the hang of the things and learn to correct your mistakes . . . and you'll make a lot of them. Programming is difficult only because it is exacting – a computer does not forgive errors, it just compounds them for you. It takes a lot of time and patience to beat new trails . . . so it is fortunate that user groups are proliferating to provide exchanges of programs . . . why keep on inventing the wheel, right?

I hope I haven't worried you about computers . . . they are not very expensive these days . . . are getting cheaper . . . and are an enormous amount of fun to play with. Get cracking . . . and as you conquer new territory, make a chart for the rest of us and send in the information to 73.

. . .W2NSD/1

by
George R. Allen W1HCI
80 Farmstead Lane
Windsor CT 06095

What's a Computer?

Many experimenters are reticent to purchase and build a microcomputer system, even though complete systems can now be purchased for less than $100. This hesitancy on the part of interested experimenters can in most cases be attributed to several factors:

1. Temporary depletion of pocket cash.
2. Lack of knowledge of computer fundamentals.
3. Lack of personal confidence in being able to handle the technology required.

I have prepared a series of articles addressing problem areas 2 and 3 (problem 1 is only temporary!) by giving the experimenter a fundamental overview of computer principles. The various components of fundamental computer systems will be discussed, computer terminology will be explained, and fundamental, inexpensive breadboard circuits and experiments will be given in order to teach the rudiments of computer technology. The simple circuits and related experiments will give the experimenter the experience and confidence needed to build and debug computer circuitry.

The average experimenter with a basic knowledge of electronics who studies these articles and performs the experiments given should be capable of building and using his own microcomputer system.

What Is a Computer?

A computer is a device which accepts information, applies some prescribed process to that information, and supplies the results.[1] This definition can be applied to large classes of devices. For example:

1. A series of gears, shafts, axles, cables, etc., such as a speedometer, takes rotation of axle (accepts information), converts the information to usable form (applies prescribed process), gives reading of speed on dial (supplies results).

2. A frequency counter takes input pulses (accepts information), counts them (applies prescribed process-counting), displays frequency on an indicator (supplies results).

3. An amplifier takes a small voltage (accepts information), amplifies it (applies prescribed process-amplification), gives larger voltage as output (supplies results).

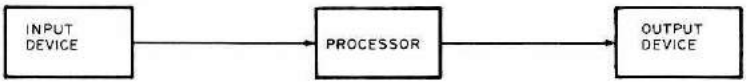


Fig. 1. Fundamental computer.

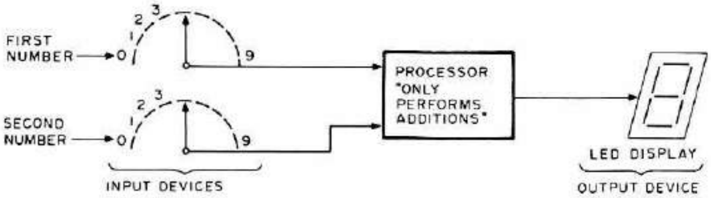


Fig. 2. Simple "addition only" computer.

These three devices are all examples of "common" computers.

A "computer" is not always recognized as a "computer," and a "computer" is not always called a "computer." The term "computer" is a broad term and may be applied to common everyday devices. Computers need not be electronic, but may be mechanical, hydraulic, pneumatic, or perhaps biological.

What is a Digital Computer?

Computers are divided into two common classes: analog computers and digital computers. Both classes of computers are the same in that they accept information, apply a process to that information, and deliver results; however, they differ in the types of information which they can handle.

An analog computer processes information within a continuous range or within continuous ranges. Using the amplifier as an example, consider a simple device which will deliver a gain of precisely 100 to voltages in the range of .03 V to .08 V. In this amplifier an input of .03 V will give 3 volts out. An input of .039927 V will give 3.9927 volts out. This simple analog computer will operate with *any* voltage within its specified range. Furthermore, *all* values of voltage within the range of the device will be processed. The range of the information that the analog computer will handle is continuous – there are no gaps within the range.

A digital computer can process only discrete values (for example, in the range 1-5,

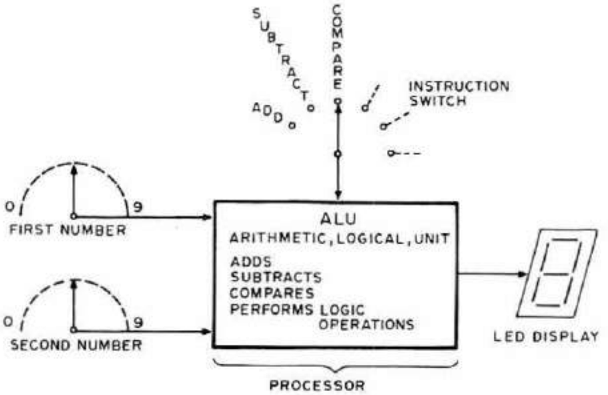


Fig. 3. Simple computer using ALU.

the numbers 1, 2, 3, 4, 5 are discrete values). The digital computer is not capable of handling continuous information. The reason for this is simple. The digital computer uses a series of "on-off" conditions to store information. The number "one" might be represented by an "on," while the number "zero" might be represented by an "off." But what about numbers such as .5? Can you have half of an "on" or half of an "off"? Certainly not very conveniently. Of course we could change our definition and let .5 be represented by an "on," but then how do you represent .55, .51, and so on? The point is that a digital computer can only handle discrete numbers, regardless of how we define those numbers. It cannot handle *all* numbers in a given range.

The frequency counter is an example of a digital computer. It accepts discrete pulses, counts them, and displays the results. In a given one second interval it cannot count 1/2, or 1/3, or .40497 of a pulse. It can count only discrete pulses.

What are the Components of a Digital Computer?

The typical digital computer may have numerous components with a rat's nest of interconnections; however, a fundamental digital computer requires only three pieces: an input device, a processor and an output device (Fig. 1).

The input device may be as complex as a graphics input terminal or it may be as simple as a single switch. The output device may be as complex as a video display or it could be a single small indicator. The processor could be very sophisticated, or it could be simple logic used to detect the simultaneous presence of switch closures.

As an example, consider a simple computer which has the sole function of adding two numbers in the range 0 to 9 together and displaying the output on an LED indicator. The block diagram of this simple computer is shown in Fig. 2.

This simple computer is a fixed function computer and can do only one function – add. The limitations of this computer should be apparent. It has a small range (0-9), cannot perform other arithmetic functions, and cannot compare two numbers for equality.

We could expand the capabilities of our processor by exchanging the simple "addition box" for an ALU (arithmetic, logical unit). This ALU type of processor is a readily available unit and offers additional capabilities such as subtraction, comparison of two numbers for equality, and logical operations such as "and" and "or." The simple configuration has now been expanded to that in Fig. 3 by the addition of an extra switch, an "instruction" switch, and by using an ALU for the processor. By varying the position of this switch, the various "instructions" could be selected. We could perform any of the allowable operations on our two "input" numbers and have the results displayed on the LED indicator. At this point, the fundamental computer has additional capabilities, but still does not have enough capability to be really practical. The ALU by itself can only process two independent numbers at any given time. It is not capable of simple steps such as adding a column of ten numbers, let alone complex problems involving many steps.

If our problem was to add a column of ten numbers, we could expand the fundamental computer still further by adding some device to store the column of ten numbers. The device could be connected to the processor in such a manner that the ten numbers would automatically be added. This storage device could be in the form of ten sets of switches, a tape recorder, a rotating magnetic disk, a series of magnetic cores or a series of electronic storage locations. A storage device in one of these classes is commonly called a memory. Note that a memory can be of several forms and is not limited to magnetic core or electronic storage. (Note: One of the earliest digital computers used a tank of liquid mercury as a delay-line memory.)

At the risk of appearing to go on and on forever, one last addition will be made to the fundamental computer system – an instruction memory. This instruction memory will serve to hold a series of steps for the processor and will give these instructions to the processor in sequence. A clock (in this example, part of the instruction memory) is used to generate pulses to step the instruction memory from one instruction to the next (Fig. 4). With this system we could command the processor to perform the following steps in sequence:

1. Add the first two numbers.
2. Add the last two numbers.
3. Compare the two sums.
4. Display the smallest sum.

Of course the sequence of commands could be endless. This simple computer system has a lot of versatility and could be very useful. (An example of this type of "programmable" computer is a programmable calculator.) The process of setting up the instructions for the computer is called *programming.* The computer as it follows the programmed instructions *executes* or *runs the program.*

A Real Life Computer

A real life computer does not differ much in logic or function from that shown in Fig. 4; however, the ALU is usually expanded to provide additional capabilities, and additional circuitry is usually provided to simplify input/output as well as to facilitate

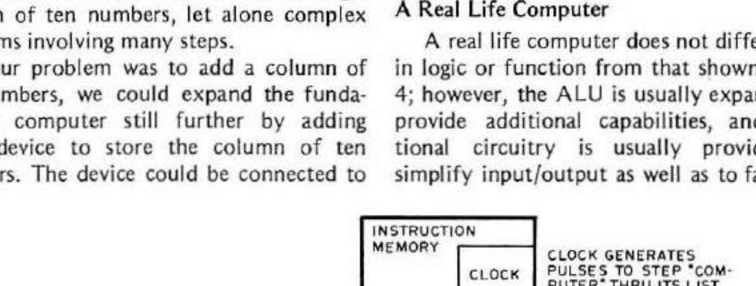


Fig. 4. Computer with memory.

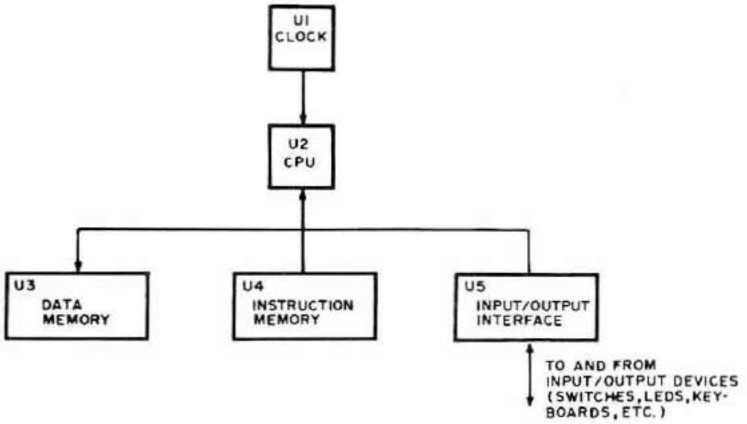


Fig. 5. "Real life" computer. U1 = 4201 clock chip; U2 = 4040 CPU chip; U3 = 4002 random access memory; U4 = 4308 read only memory; U5 = 4207 (4209, 4211) general purpose I/O.[2]

the flow of information and instructions within the system. While this last statement may sound like a "zinger," it is not, since a device called a "CPU chip" (a single integrated circuit) in most cases contains the additional circuitry as well as the arithmetic and logical functions. A CPU chip (Central Processor Unit) is a very unique and versatile device and is commonly called a "microprocessor." Fig. 5 shows a block diagram for a typical, fundamental microprocessor computer system. The five blocks shown correspond to 5 integrated circuits. This is a real life system. An Intel 4040 microprocessor system could be built using just 5 ICs.

How Do I Get Started?

It is not difficult to wire five integrated circuits together to form a microprocessor system. It is difficult, however, to make the plunge without first acquiring some important fundamental knowledge. By gathering this basic knowledge first, you can better utilize your microprocessor and you are in a better position to correct a problem should difficulties be encountered.

Fortunately, a large investment is not required in order to get into computers. For a nominal investment in a power supply, a breadboard, and a handful of very inexpensive ICs, the experimenter can build up simple computer-oriented circuits to experience firsthand just how things work. While simple circuits will not duplicate all of the functions of a microcomputer system, the experimenter will be able to perform arithmetic and logical operations as well as store and retrieve information with simple memories. If the experimenter is able to understand these basic concepts and is able to duplicate simple experiments, then he should be able to build and use a microcomputer system. Future articles will explain some of these simple circuits and experiments.

The Next Step

Man works with numbers in the decimal system, while computers work with numbers in the binary system. Future articles will discuss number systems, will explain how to do arithmetic in the binary system, and will give procedures to convert from one number system to another. Several experiments have been designed using an inexpensive ($3.95) ALU chip to illustrate binary arithmetic. Future articles will also discuss logic, memories and input/output. ■

References

[1] ***Computer Dictionary and Handbook,*** **C. J. Sippl, C. P. Sippl, Howard W. Sams & Co., Inc., 1972, page 99.**

[2] ***MCS-40 User's Manual For Logic Designers,*** **Intel, Santa Clara, CA, 1974, pages 3-4.**

WHERE COMPUTERS ARE AT

Those readers of *73* who are into microcomputers are in on the ground floor of something new and exciting. I don't think many people realize how incredibly new the low cost computer field actually is . . . in point of fact it just got started in 1975 with the introduction of the MITS Altair 8800 processor.

By way of comparison, a computer system which might have cost $1,000,000 in 1970 was down to about $100,000 by 1974 by virtue of the application of ICs to computers. These medium priced systems were called minicomputers . . . they were mini in size, but their capability wasn't much different from most of the earlier maxis. Then came the computers-on-a-chip . . . the Intel 4004, the National IMP-16, the Intel 8008, the 4040 . . . and finally the Intel 8080. Motorola came up with the M6800, Fairchild with the F8, National with their PACE, General Instrument with a C1600, and MOS Technology with the 6501.

The micro chips revolutionized the computer industry again, resulting in the microcomputer or microprocessor, as it is often called. As with the mini, only the name and size had changed . . . the job these micros could handle were about the same as the bigger computers . . . but now the price for a system had dropped from $100,000 to about $10,000.

Have we reached the end of this progression? Not by any means. The chip people are still busily at it, designing cheaper and cheaper microprocessors . . . the latest is the $25 6501 chip which is a direct replacement for the much more expensive M6800 . . . and, as engineering costs are amortized, we can look for these costs to drop further.

The cost of peripherals will also drop as firms gear up for mass production of interface and control chips which will replace the IC-laden circuit boards of today. One-chip video display generators will probably bring the cost of the video keyboard down to the $100 range . . . like a black and white television . . . though that may take three or four years. Right now anyone that comes out with a $400 video keyboard will make millions of dollars . . . until the $350 unit comes out.

None of the present day tape storage systems are ideal for small computer systems and the race is on to invent a mass memory storage system which is geared to the low cost computer. Small businesses and homes don't need very large storage systems, nor do they need lightning fast service, so a tape system which would allow you to get your data in a few seconds would cut the mustard . . . a good trade off against price.

There are fantastic opportunities in the small computer market for making large gobs of money. There will probably be more computers sold in the next year than there are in existence in the world today . . . and maybe five to ten times that many in the next year. The hobbyist of today who gets to know the field and is able to take advantage of what he knows could be the mogul of tomorrow.

By way of getting used to how early we are in the field, as of late summer 1975, to my knowledge, even MITS had not yet started shipping complete computer systems with floppy disk operating units . . . they were close to it. Remember that MITS, as of this date, was the *only* supplier of micro systems in the country. Several others were about ready to go. By the time this reaches print I expect that there will be at least three, perhaps four, firms supplying operating microcomputer systems . . . hopefully with working floppy disk memories. Floppies will have to hold until one of the mentioned tape systems is invented. Floppies are okay . . . a little limited in capacity . . . and expensive by hobby standards . . . $1000 to $2000 range.

HAMMING IT UP ON COMPUTERS

How will we be using computers in amateur radio? Will they further reduce the quality of contacts, making them even more mechanical than today . . . if that is possible?

I realize that it is popular to bad mouth developments such as this, but the fact is that my experience indicates that it may well be that amateur radio never had it so good. Those of us who have delved into radio Teletype communications are not unaware of the fascination that this mode holds. We also are probably aware of some of the reasons for the excitement and fun of RTTY as compared to CW and SSB.

There are some basic problems involved with radiotelephone communications to which we have given too little consideration. After spending our childhood years learning how to talk with people we can see and hear . . . getting feedback from them as we talk in the way of grunts, yesses, nods, etc., and even getting these during telephone conversations . . . we are unprepared to handle the prospect of talking to someone we can neither hear nor see. We are at a loss for words . . . known as mike fright in beginners . . . known as incredibly boring contacts and deadly routines in older timers . . . the 10-4 of the CBers . . . the endless repetitions of some amateurs . . . the meaningless descriptions of stations, weather, and such.

Most of your life patterns are well set by the time you are five, and that includes your ability to talk to people you can't see or hear. The result is that the brain boggles and rational conversation is virtually impossible. This may explain the difference between talking to hams in person and trying to work up anything more than an exchange of pleasantries over the air.

Continued on page 11

John Isaacs W6PZV
3175 Val Verde Ave.
Long Beach CA 90808

Number Systems

-- a brief history

Have you ever wondered about the term "binary number" that you have seen in electronics magazines? Since we already have a perfectly good decimal number system, why complicate things with another? If you have a few minutes, here is your chance to learn more about number systems than you ever wanted to know.

One of the most important achievements in the development of science has undoubtedly been the invention of our decimal number system. Counting in units of ten must certainly be due to the fact that man has ten fingers. In some cases, some people have counted in units of five or twenty, which correspond to the use of one hand or of both hands and both feet.

The main use of numbers in early times was for simple counting and record keeping. The numeration methods were designed chiefly for those purposes. With the development of trade and the sciences, the numeration became more and more inadequate. It took a long time before an adequate number system was devised. The Greeks and the Romans did not succeed in this endeavor even though they achieved a rather high development in science. Just imagine performing simple arithmetic with Roman numerals, like dividing MMDXLVI by CCIX using Roman numerals only. About six hundred years ago any simple operations like multiplication and division of large numbers required the services of an expert.

The Hindus and the Arabs are credited with the concept of positional value and the use of the zero, which are explained as follows. Consider the number 74638. We understand this to mean 7 x 10,000 + 4 x 1,000 + 6 x 100 + 3 x 10 + 8 x 1. In the following numbers, the 4 has different values depending on its position: 42,000, 240, 324. In the first number 4 is equal to 40,000, in the second its value is 40 and in the last just 4. Without a 0 to place in some of the positions, how would you write three hundred and six if every position had to have a number? Use of the concept of zero enables us to write 306.

The positional values of the base ten number system are as follows:

$$10^3\ 10^2\ 10^1\ 10^0.\ 10^{-1}\ 10^{-2}\ 10^{-3}$$

You will note that in each case the value of the position is the radix (10) raised to some power. It is necessary to go to the right of the radix point (decimal point) in order to express fractional numbers.

The general expression for a number is then:

$$N = a_1(r)^{n-2} + a_2(r)^{n-3} + a_3(r)^{n-4} + a_4(r)^{n-5}$$

In the above expression r is the radix or number base and the a values are the position numbers. In order to have a base ten number system that will translate most practical numbers we would have to make n = 5. Using the arbitrary number 2307.602 as an example, and making n = 5, we derive:

$$2(10^3) + 3(10^2) + 0(10^1) + 7(10^0) + 6(10^{-1}) + 0(10^{-2}) + 2(10^{-3})$$

Note that any number raised to the zero power is equal to one (i.e., $10^0 = 1$).

Now we can calculate the positional values for the binary (base 2) number system. They are as follows:

2^0 =	1
2^1 =	2
2^2 =	4
2^3 =	8
2^4 =	16
2^5 =	32
2^6 =	64
2^7 =	128
2^8 =	256
2^9 =	512
2^{10} =	1024
2^{-1} =	.5
2^{-2} =	.25
2^{-3} =	.125
2^{-4} =	.0625
2^{-5} =	.03125
2^{-6} =	.015625
2^{-7} =	.0078125
2^{-8} =	.00390625
2^{-9} =	.001953125
2^{-10} =	.0009765625

We now have the conversion factors necessary to convert a binary number to a decimal number.

Let's consider the base 2 number 0011100.0 and calculate its decimal equivalent.

$$\begin{array}{l} 0 \times 1 = 0 \\ 0 \times 2 = 0 \\ 1 \times 4 = 4 \\ 1 \times 8 = 8 \\ 1 \times 16 = 16 \\ 0 \times 32 = 0 \\ 0 \times 64 = 0 \\ \hline \end{array}$$

28 (base 10) = 0011100.0 (base 2)

It may not have been apparent from the preceding, but the fact is that binary numbers are made up of 1s and 0s only. If we were to design an electronic computer or calculator to use the decimal number system, we would find that the decimal number system was capable of the high speeds that are necessary — but it would be very difficult to provide ten stable states so that the computer could handle the decimal numbers. Fortunately, any number can be represented in the binary number system, so computers are designed to use it. They might also use the base 8 or base 16 number systems, since these have similar properties.

Electronic circuits have been devised that will add, subtract, multiply, and divide in the binary number system. If you are interested in these processes you could consult any text on computer design. Several are listed in the references at the end of this article.

Just to stimulate your interest in the binary number system, let's try working a problem. First we need to know that in the binary system 1 + 1 = 0 plus a carry. The carry is into the next position (or 2^1) so that properly 1 + 1 = 10. This happens in the decimal system also; for example, 9 + 9 = 8 plus a carry into the 10^1 column, so that 9 + 9 = 18. Now let's try out your new understanding of number systems on this little problem. Calculate the first five positional values to the left of the radix point in the base 8 number system (remember, $8^0 + 1$). Then calculate the decimal equivalent of the octal number 40. The answers follow the references.

Now that you are up on number systems, you don't have to skip over the articles that refer to the binary number system. Things like electronic keyers use them, especially if there is a memory involved. A lot of the integrated circuits manipulate them also. Have fun! ■

References

Hallerman, *Digital Computer System Principles,* McGraw-Hill.
Hawkins, *Circuit Design of Digital Computers,* John Wiley & Sons.
Lebow and Reed, *Theory and Design of Digital Machines,* McGraw-Hill.
Ledley, *Digital Computers and Control Engineering,* McGraw-Hill.

● Answers: 1, 8, 64, 512, 4096; 32.

I/O Editorial

From page 9

With RTTY we find that we have a lot more time to engage the brain and give some thought to what we are trying to say. The result is that in general our communications are a lot more interesting . . . more like a series of letters being exchanged than talking. It is just possible that amateurs using computers for contact would go this route. They probably would even go one step further and keep a tape of their better efforts so they would be available during later contacts.

Just imagine what an interesting story you could work up for a contact about a recent trip . . . a DXpedition . . . or some good DX worked . . . some adventures in putting up your tower . . . stories about how a beam works . . . you could put real life into your contacts.

Let's move the clock ahead and imagine a contact in 1978 . . . it is not your computer working his computer, but it is something like that. You would get in touch and your screen would have the hello and a list of some things you might be interested in talking about . . . perhaps a list of some things you might be interested in talking about . . . perhaps a list of some stories he has on tap for you. You give an okay to the tower story and immediately your screen comes up with the tale of how he read about this tower . . . got the parts to build it . . . got it all together and then couldn't get it raised! Your screen, when you give the keyboard a "go" gives you a couple pictures of the chap and his friends trying to raise the tower on slow scan. Then more story when you give him another "go."

Does your friend have to sit there and wait while you do all this? No, of course not . . . he has gotten the same type of stuff from your system and is busy watching the story and pictures you have on tap for him.

Won't all this take up a lot of the band? Probably not . . . the chances are that hundreds of contacts will be carried on in short bursts on a single channel . . . possibly with a fixed frequency receiver . . . or it may come down to several fixed channels such as on FM today.

On the other hand, it might not work out like that at all . . . but it could.

COMPUTERMANIA

Despite a whole lot of effort, I am still an utter neophyte in the computer field. I want to establish that before I tell you about computers. Okay?

Now that I am no longer involved with *BYTE* magazine, but still haven't lost my enthusiasm for computers and their future in Amateur Radio, I'd like to do everything I can to encourage them's as what knows to write for us'n's which wants to know . . . for *73*.

Now before you start to write, you computer expert, you . . . I want to lay on you a little thought . . . the great unwashed of us out here don't even speak your language yet. We don't know a compiler from an assembler. Oh, we really want to know and we're looking for you to explain the whole works to us in simple terms which we can understand, not in computer jargon. Just take a look at the beautiful job Larry Kahaner has been doing in explaining the basics of digital design and you'll get the idea of what we need.

We want to know about computer design, ham applications of computers, the differences between the various systems such as the 8080 based processors, the 6800 based, PACE based, etc. Sure, we know that in the end all of them will do just about anything we want them to, but we also want to try and understand what the differences are and what these differences mean to us.

We want to know about how to use our teletype machines . . . our television typewriters . . . our cassette recorders . . . how to interface other computer equipment and use it . . . how good or bad some of the new hobby computer equipment is and what we can or can't do with it . . . stuff like that.

We want to get a general idea of what the major computer languages mean, why which are used and when, which we should learn and why, and so forth. We want to have an understanding of what people mean when they refer to Fortran, to Basic, to Cobol, to RPG, to PL, and other common program languages.

We want to be able to hook up computer systems and use them . . . for games . . . for hamming . . . perhaps for printing out the schedule of Oscar 7 or the times of possible acquisition of Oscar for active Oscarites . . . moonbouncers want to know where the moon is or even have a computer point their antenna . . . and who of us would not like to be able to keep track of all our ham contacts in a computer system?

We want to know where we can get programs for use in our computers, from the simpler systems such as the HP-65 on up to Altair 8800, Sphere 6800, SWTPC 6800, Godbout PACE, etc. The hobby computer field shows no sign of slowing in its growth.

So there you have it . . . we need articles. Writing for *73* is simple . . . double space type it with generous margins, a sketch of diagrams will do (but be very careful to include everything possible), and the best photos possible. If you have trouble with photos send the stuff here and we'll take 'em. If you have a source of PC boards for construction projects be sure to let the readers know . . . if not we'll try to find one. We also like to have a negative of PC boards available for interested readers if possible.

The best part, perhaps, is that we pay and pay generously for articles. Not only are you able to share your work with others and bring them fun and happiness, you also get paid . . . and you get well known, too. Ask Pete Stark and other regular *73* authors about that! If publishing will help you in your work or to get a better job, this is a good medium. Of the ham magazines which pay for material, *73* has by a wide margin the largest circulation, no matter what you may have read in the way of half-truths (well, 39.5% truths) elsewhere. You want your articles to get the widest distribution, don't you?

Get busy, then, and teach us something about computers . . . and try to remember what it was like to not know computer language when you write for us.

Silas S. Smith, Jr. WA9VFG
2308 McCord
Murphysboro IL 62966

Is Digital All That New?

To meditate on an experiment, a serious experimenter often shelves today's experiment, perhaps never to take it up again. The "light" was in the meditation and not in the experiment. This brings me to the point at hand — meditation as related to digital.

For several years now, a transition from analog to digital has been in the process. To many people, digital is considered as a rather new development, but I ask, "Is it really all that new?" The question may suggest the answer that digital has been around for some time.

Just for the fun of it, let us take a brief and somewhat abstract view of digital devices or communications through the age of man.

From man's beginning, he had a very good calculator at his fingertips as well as a computer with all the software that is necessary for its operation, although he had very little need for either in his primitive culture. It was 2 to 3 million years before man saw the need to communicate between computers. Man has, over the last 20,000 years, developed ways to communicate between computers.

If the old axiom is true that a picture is equal to a thousand words, and if we should equate the picture to a computer address and equate the thousand words to the memory of the computer, then the cave man had the basic idea in his cave drawings some 20,000 years ago. Programming his computer to accept more addresses (more pictures), he was able to develop a memory system called hieroglyphics, which used approximately 900 pictures. Up to this time man had almost exclusively used analog, which is called the spoken language. Hieroglyphic programming was too complex for most computers, so a much simpler system was developed using only about 24 pictures. Hierotics, as this system is called, was very effective digital-wise, but lacked the ability to communicate with analog systems. About 3,000 years ago a digital to analog and analog to digital system was developed. This system is often called phonic writing. Soon to follow was a digital system that was much more simple and was easily converted to analog. This system was made by the combining of the Phoenician phonic system and the Roman alphabet. Because this system had only 26 digits or letters and could be formed into word groups, it was adaptable to programming. Such programs are placed into a machine called a printing press and are read out into a permanent storage unit known as a book, which is a type of read only memory. With this type of ROM, the 2 or 3 million year old computer design has almost unlimited capacity to perform the most complex problems.

In Aristotle's theory of the universe formulated sometime around 400 BC, it was suggested that a binary system was possible. Such a system would consist of such variables as hot and cold, wet and dry, light and dark, etc.

Much experimenting in digital communications was taking place by the mid-1700s, parallel versus serial as each form was being developed. Static or friction electricity and its pith balls came first as a parallel system (i.e., one set of pith balls for each letter). Soon to follow was a synchronized serial form using only one set of pith balls.

Galvan electricity and the use of magnetic needles went through the same parallel and serial development. However, a new wrinkle was added. The new wrinkle was code — that is, left or right, operated or non-operated. Its form was much like that of Morse code.

From its inception in the mid-1800s, the electromagnetic (telegraph) system used a code in serial form. It was the search for the conversion of the electromagnetic digital system to analog that led to the development of the telephone — which just happens to be an analog device. It was only a few years ago that the conversion of digital to analog and analog to digital was electronically accomplished. Its form is called pulse code modulation.

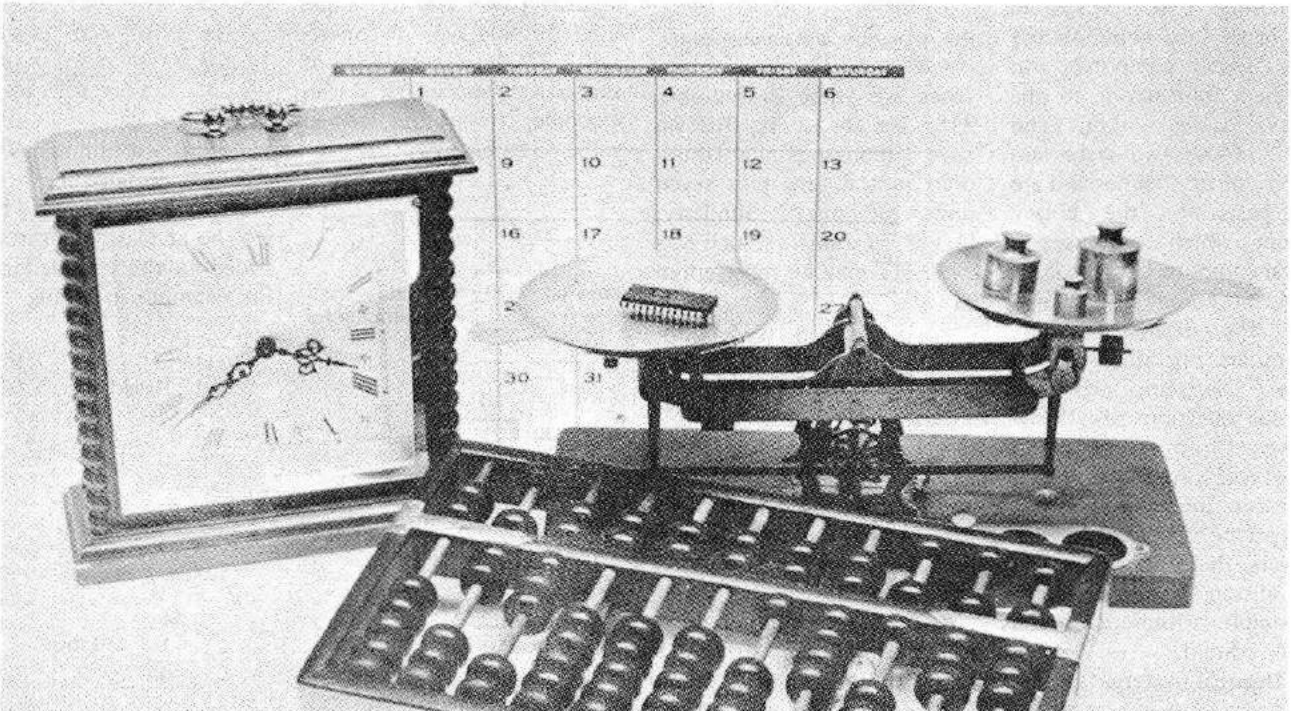
Just as in the addressing of electronic computers, we have been simplifying the addressing to the 2 or 3 million year old design. Some examples include changing "The United States of America" to USA, and other conversions resulting in terms such as IRS, FCC, IBEW, NAACP, FBI, CIA, 73 and others.

What is the answer to the question, "Is digital all that new?" I would say no. I would go so far as to say that at present we are not really in a truly transitional period but rather in a sort of jockeying position. As we turn the corner, I predict that the next big development will be a 3D computer that will make the present computer look like a small contribution to the overall communications system — but I see no way that the old 2 or 3 million year old design will ever be replaced. ■

George R. Allen W1HCI
80 Farmstead Lane
'Jindsor CT 06095

Two Finger Arithmetic

--how computers figure



We are not aware of all the number systems we use daily, as represented by the items shown here.

The first caveman, when he learned how to count on his fingers, gave us the decimal number system which we use today. While this number system is second nature to us, it is not the only number system with which we come into contact every day. Timekeeping, for example, uses a unique numbering system which is based on the numbers 60 and 24. There are 60 seconds in a minute, 60 minutes in an hour, and 24 hours in a day. And of course, the English have blessed us with unique number systems for weights and measures — who can forget that 4 gills = 1 pint, 2 pints = 1 quart, and 4 quarts = 1 gallon? If you stop to think about it, you can see that we are involved with many number systems other than the decimal system.

The advent of the computer age has ushered in an additional number system, the binary system based on the number two. This system has come into common use since digital computers can represent information in one of two states — "on" and "off." These two states are called "binary" states and are the basis for the binary system which we use in digital computers.

Man commonly works with the decimal system, computers operate with the binary system, and the obvious questions are "How do we get from one system to the other?" And, "How do I represent information in a computer?" This article answers these questions and in addition explains how to do simple arithmetic in the binary system.

The Decimal System

We are constantly thinking numbers, adding numbers, and writing down numbers; but, do we really have an understanding of what we are doing? When we write down a number such as the number 3187, what does it really mean?

All numbers are made up of a series of digits. A number can have as few as 1 digit and is not limited to any maximum number of digits. In the decimal system, each place occupied by a digit has a power of ten associated with that place. For example, in the number 3187, we have four places. The powers of 10 associated with the four places are as follows:

digit	3	1	8	7
power of 10	10^3	10^2	10^1	10^0

The number 3187 could be written as the following sum:

$$3187 =$$

$$3\times10^3 + 1\times10^2 + 8\times10^1 + 7\times10^0$$

and if we remember our basic mathematics we will recall that

$$10^3=10\times10\times10=1000$$

$$10^2=10\times10=100$$

$$10^1=10$$

$$10^0=1$$

(By definition, *any* number to the zero power = 1.)

When we write the number 3187 we are saying that we have 3 thousands plus 1 hundred plus 8 tens plus seven units (or ones). Similarly, larger numbers such as 5197283 may be represented as

$$5\times10^6 + 1\times10^5 + 9\times10^4 +$$

$$7\times10^3 + 2\times10^2 + 8\times10^1 +$$

$$3\times10^0$$

The Binary System

Computers operate with the binary system because each digit can have only one of two states — "on" and "off." Numbers are represented in a computer by a string of binary digits called "bits." Consider a number represented by 4 binary digits (bits) when off = 0 and on = 1. Four lights may be used to display the "on" and "off," or one and zero respectively:

number	on	off	on	on
numerically	1×2^3 +	0×2^2 +	1×2^1 +	1×2^0

We know that

$$2^3 = 8 \qquad 2^1 = 2$$

$$2^2 = 4 \qquad 2^0 = 1$$

Our number represented by the lights above would be

$$1\times8 + 0\times4 + 1\times2 + 1 = 11_{10}$$

Instead of actually writing down lights to indicate the "on" and "off" states, we use the binary numerals 0 and 1. Thus in base 2 our number is written as 1011_2. If it is very clear that you are working with binary numbers, you may omit the subscript 2.

Conversion from Decimal to Any Base

Conversions from a given base to the decimal system can be made by expanding a number to the powers of its base (as shown in the previous section). But, how do we take a decimal number and convert that number to another base? The technique used for this type of conversion is the technique of successive remainders. As an example, convert the decimal value 43 to base two:

$$\begin{array}{r} 21 \\ 2\,\overline{)43} \\ \underline{42} \end{array}$$

remainder 1 is the multiplier for 2^0

$$\begin{array}{r} 10 \\ 2\,\overline{)21} \\ \underline{20} \end{array}$$

remainder 1 is the multiplier for 2^1

$$\begin{array}{r} 5 \\ 2\,\overline{)10} \\ \underline{10} \end{array}$$

remainder 0 is the multiplier for 2^2

$$\begin{array}{r} 2 \\ 2\,\overline{)5} \\ \underline{4} \end{array}$$

remainder 1 is the multiplier for 2^3

$$\begin{array}{r} 1 \\ 2\,\overline{)2} \\ \underline{2} \end{array}$$

remainder 0 is the multiplier for 2^4

$$2\,\overline{)1}$$ (doesn't go)

remainder 1 is the multiplier for 2^5

Our number expressed as powers of two is $1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$ or 101011_2. Checking ourselves and converting back to decimal, $1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 32 + 0 + 8 + 0 + 2 + 1 = 43$.

Binary Arithmetic

Arithmetic in the base two is very simple. You don't have to remember a lot of arithmetic facts; all you have to remember is zero + zero = zero, one + zero = one, and one + one = ten. As an example:

$$\begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

With binary arithmetic, whenever two or more ones appear in a column to be added there will be at least one carry bit added to the next left digit. For example, in adding

$$\begin{array}{r} 11 \\ +\;\;1 \\ \hline \end{array} \qquad \text{right digit} \quad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

carry bit ↑

we get

carry → 1

$$\begin{array}{r} 11 \\ +\,1 \\ \hline 0 \end{array} \quad \text{and then} \quad \begin{array}{r} 11 \\ +\,1 \\ \hline 100 \end{array}$$

Storing Numbers in a Computer

When we talk about numbers stored in a computer, we don't normally speak of them in terms of bits (binary digits); we usually speak of them in terms of bytes or words. A byte by definition is a collection of sequential bits. A byte can be any number of bits but is most commonly 8 sequential bits. A computer word is a collection of bytes and is defined as the number of bytes pointed to by one addressing operation in a computer. Words and bytes tell us about the organization of the computer. As an example, assume the standard definition of byte, where byte = 8 bits:

> The XDS Sigma 6 computer is a "32 bit" computer; it has 32 bits per word. 32/8 = 4, thus there are 4 bytes per word.
>
> The DEC PDP 11 series of computers have "16 bits" per word. It has two bytes per word.
>
> The INTEL 8080 microprocessor is an 8 bit microprocessor; it has 8 bits per word. Thus it has one byte per word.

The term byte is used very frequently as the definition for a unit of information. Alphanumeric characters (letters, punctuation, printable characters, etc.) are usually stored in coded form in "one byte."

Numbers written on paper are stored in visual form and for all practical purposes there is no limit to the size of the number on paper. In a computer, there are limits set, due to the "word size" and due to the arithmetic capabilities of the computer. In large scale computers, it is possible to work with very large numbers such as those which can be represented in 64 bits or more. In microprocessors, however, the limit is very small, usually being 8 bits but in some cases being 4 bits. The arithmetic capability that we are talking about is the type of arithmetic which can be performed in one computer instruction. It is possible to string instructions together and thus handle larger numbers. This is usually done by "software."

Computer Arithmetic

As previously mentioned, computers have limits set on their arithmetic capabilities. They cannot in a single instruction perform simple arithmetic on all size numbers; they are limited to performing arithmetic on some given number of bits. Because of these limitations, computer arithmetic is somewhat different from binary arithmetic. Computer arithmetic is binary arithmetic within limits.

A typical microprocessor, such as the Intel 8080, has 8 bit arithmetic capability. It can perform arithmetic on 8 bit numbers. It cannot in a single instruction operate on a 32 bit number.

The largest number that can be expressed in "8 bit arithmetic" can be determined as follows: Consider an eight bit binary number such as 11111111_2, all one bits in the eight bits. The number can be expanded as $1x2^7 + 1x2^6 + 1x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0$, which is the same as $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255_{10}$. This is the largest number that can be stored in a single 8 bit word (or byte) and also the largest *sum* that can be accumulated from the addition of two numbers. We can add $250_{10} + 5_{10}$ and get a sum of 255, but we cannot add 250_{10} and 6_{10} to give 256_{10}, since that sum is beyond the arithmetic capabilities of 8 bit arithmetic. Of course this arithmetic restriction only applies to a computer with 8 bit arithmetic. If the computer used 4 bit arithmetic or 32 bit arithmetic, then the actual largest number would be different. In a four bit arithmetic microprocessor, the largest number would be $1111_2 = 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0 = 15$.

In order to simplify the examples, all discussion in this section will pertain to a microprocessor or computer with 8 bit arithmetic capability.

If our microprocessor can handle eight bit arithmetic and can also store sums up to 255, then what is to prevent us from attempting to add two numbers that will produce a sum greater than 255? The computer doesn't know what the sum will be before the addition, so we can at least make an attempt. The answer is, "Yes, you can instruct the computer to perform the addition, but the results will be wrong." An *overflow* condition will result. The storage capabilities and the arithmetic capabilities of 8 bit arithmetic will be exceeded. What happens is analagous to trying to pour 3 quarts of water and 4 quarts of water into a five quart container. The container will overflow.

Subtraction and Negative Numbers

Most microprocessors available today have subtraction capabilities; however, the microprocessor user may or may not wish to use those subtraction capabilities, depending on the capabilities available. The user may wish to "complement" the number to be subtracted and then perform an addition, so that the arithmetic techniques may be simplified. Two types of computer arithmetic will be discussed in this article – ones complement arithmetic and twos complement arithmetic.

Both negative numbers and subtraction are used in a computer, but not necessarily in the same manner as with pencil and paper. In order to indicate a negative, instead of writing down a minus sign on paper a "sign bit" is set to "one" somewhere in the computer. This "sign bit" may be stored in a word by itself or it may be stored in the same word in which the number is stored. Most commonly, the "sign bit," which indicates a negative number, is stored with the number itself in the leftmost bit position of the word. In an eight bit word, the sign bit would be as shown:

Sign bit ⓧx x x x x x x

In a sixteen bit word the sign bit would still be the leftmost bit as:

Sign bit
↓
ⓧx x x x x x x   x x x x x x x x

If the sign bit is a "0", then the number stored in the word is positive. If the sign bit is a "1", then the number stored in the word is negative.

The maximum range of numbers which can be represented in an eight bit computer would be as follows:

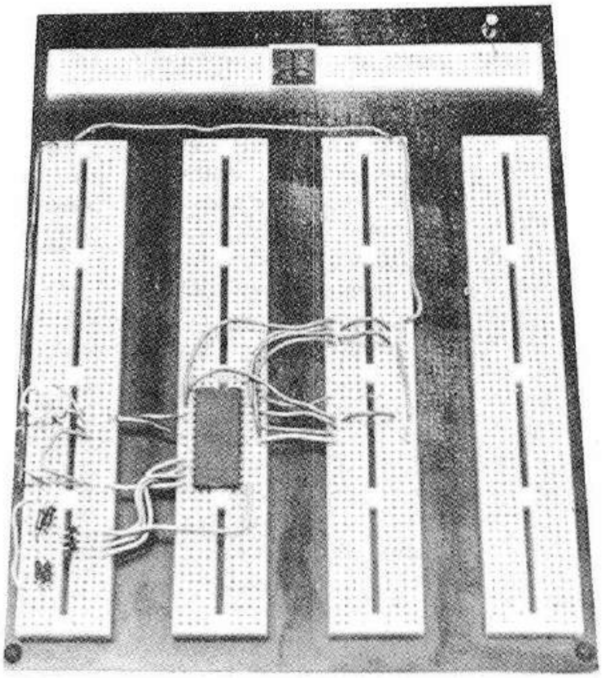
Sign bit
↓
0 1 1 1 1 1 1 1 = 127_{10}
(Most positive value)
0 1 1 1 1 1 1 0 = 126_{10}
.
.
.
0 0 0 0 0 0 1 0 = 2
0 0 0 0 0 0 0 1 = 1

0 0 0 0 0 0 0 0 = ZERO

1 1 1 1 1 1 1 1 = -1
1 1 1 1 1 1 1 0 = -2
.
.
.
1 0 0 0 0 0 0 1 = -127_{10}
1 0 0 0 0 0 0 0 = -128_{10}
(Most negative value)

In an 8 bit word we have defined one bit as the sign bit, leaving 7 bits for the data. This means that the largest number including sign that can be stored in an 8 bit word is 127_{10}. This puts a further restriction on arithmetic operations as can be seen by the example, 64 + 64.

	64 in binary	= 01000000
+	64	= 01000000
		10000000



"Hands on" experience in computer arithmetic using a 74182 IC will be the subject of the next article.

By definition, we now have a negative number! We have changed the sign of our number. As can be seen, the right combination of numbers added together will not produce overflow, but will change the sign. Thus, it is also important to test to see if the sign is changed when performing addition on numbers of like sign. If the sign has changed, the 7 bit capability has been exceeded.

In computer logic, it is easy to complement a word such that all ones become zeros and all zeros become ones. This capability is usually a standard feature within the arithmetic-logical unit of a microprocessor. The complement of the eight bit binary number 01001101 would be 10110010. The number 3 represented in 8 bits is 00000011, and the complement of three would then be 11111100. This type of complement where zeros are exchanged for ones and ones are exchanged for zeros is called "ones complement."

By complementing a number using the "ones complement," we have placed a 1 bit or a sign bit in the leftmost bit of the word, indicating that the number stored in the word is a negative number. A typical ALU will complement a word internally, and then perform addition in order to effect subtraction. This might happen as follows, for the operation 7-4:

7 in binary = 00000111

4 in binary = 00000100

The ones complement of 4 (negative 4) = 11111011. Adding the two together,

$$\begin{array}{r} 00000111 \\ 11111011 \\ \hline 1\ \ 00000010_2 = 2_{10} \end{array}$$

carry bit
(overflow)

But the answer is off by one and overflow has occurred. We got an answer of 2; the answer should have been three. This is the shortcoming of ones complement arithmetic, and is the reason all modern computers use twos complement for arithmetic operations and representing negative numbers.

Twos Complement Arithmetic

Twos complement arithmetic operations can be found in all present-day computers, from the lowly 4 bit microprocessor all the way up to the giant large scale systems. The reasons will become evident as we go on and see how effectively negative numbers can be represented in this form.

To find the twos complement of a number, take the ones complement of the number and *add 1*. For example, find the twos complement of 6:

6 in binary	= 00000110
ones complement of 6	= 11111001
add 1	1
twos complement	= 11111010

This value (11111010) represents a *negative six*. Therefore, we can say that when we take the twos complement of a positive number we are changing it to a negative value. To best illustrate that this value is in fact a negative six, let us increment it *six* times (up to zero):

```
   11111010    Negative six
   +      1
   11111011    Negative five
   +      1
   11111100    Negative four
   +      1
   11111101    Negative three
   +      1
   11111110    Negative two
   +      1
   11111111    Negative one
   +      1
 1 00000000    ZERO
 ↑
Carry discarded
```

Following are some examples of *subtraction* in twos complement arithmetic:

1. 6-5

5 in binary	= 00000101
ones complement of 5	= 11111010
add 1	1
twos complement	= 11111011

Summing,

6 in binary	= 00000110
twos complement of 5	= 11111011
	1 00000001

A carry is produced, but is discarded. The sign (most significant bit) is zero, indicating the result is positive.

2. 3-3

3 in binary	= 00000011
ones complement of 3	= 11111100
add 1	1
twos complement	= 11111101

Summing,

3 in binary	= 00000011
twos complement of 3	= 11111101
	1 00000000

And, the correct answer is, of course, zero.

3. 3-4

4 in binary	= 00000100
ones complement of 4	= 11111011
add 1	1
twos complement	= 11111100

Summing,

3 in binary	= 00000011
twos complement of 4	= 11111100
	11111111

The result is negative one in twos complement (no carry produced).

In twos complement arithmetic, the addition of two numbers of opposite sign will always produce the correct result. It may or may not produce a carry. The carry or overflow may be ignored.

Notice that, in both ones complement arithmetic and twos complement arithmetic, addition stays the same. The only thing that is changed is the way in which the complement is taken in order to effect subtraction. Some ALUs will compensate automatically for subtraction in ones complement arithmetic, while some won't.

In twos complement arithmetic, the programmer should test for overflow. In any addition of like signed numbers where there is a possibility that the maximum sum could be exceeded, a test must be made for overflow. If the test is not made, there is a danger that erroneous results could occur and that the user might not be aware of that fact. In reality, any addition could produce overflow. While the user may never expect overflow to occur, if his data were erroneous, then an erroneous sum could result. By making the test for overflow under all conditions, those errors "which couldn't possibly happen" would be detected.

All of the examples given here used 8 bit arithmetic to simplify the examples, but the principles and concepts discussed hold true regardless of the arithmetic word length used. On a 32 bit machine, larger numbers can be handled, but the test for overflow (carry) and change of sign must still be made.

Arithmetic Software

Binary arithmetic within a computer is not difficult or mysterious; however, care must be given to making sure that the results obtained from an arithmetic operation are correct. The care required can add additional steps to a program and can conceivably make a simple problem into a lot of work. One way in which some of the work can be eliminated is to choose a microprocessor which provides an automatic "hardware" scheme for testing for overflow or change of sign. Another solution is to choose a microprocessor which provides the same capabilities by software. Software in this case is a program furnished by the manufacturer to do the proper testing for you. This software may also offer the capability to work with 16 or 32 bit numbers and may in addition offer other capabilities such as multiplication and division.

The Next Step

This article avoided some of the more difficult arithmetic operations, such as multiply and divide, in order to dwell on some of the fundamental concepts needed to get "on the air." Multiply and divide can wait for later. If they are needed now, the user can perform successive additions or subtractions as required, or he can use software provided by the manufacturer. These functions will be covered in a later article. It is important at this time to give the experimenter "hands on" experience with computer arithmetic so that experience and confidence can be built up. For this reason, the next article will describe simple experiments with computer arithmetic using a 74181 Arithmetic Logic Unit. These simple experiments will use the concepts discussed in this article and will help the experimenter to better grasp computer arithmetic. ■

I/O EDITORIAL

uP

That's u as in micro, a P as in processor. The world of uP is changing too fast to keep in touch. Like the first really big uP chip was the Intel 8080, which came out at around $295, as I recall, and was a bargain at that price. Bill Godbout called the other day to lay our ears back with his latest deal . . . an 8080 (prime chips, he says, not seconds or rejects . . . prime), eight of the 2102 RAM memories for 1K bytes of RAM, plus a 5204 5K byte PROM . . . the works for $65. Lordy!

That should get the rest of the experimenters off their duffs and busy putting uPs together. What can you do with 4K PROM and 1K RAM plus a uP chip? Well, you sure can run a repeater to a fare-thee-well, have an extremely sophisticated keyer with all sorts of memory, and even play a lot of the simpler computer games. The fact is you have a simple computer which can be expanded with more RAM memory and some interfaces to work with slow scan, with teletype, cassette recorders, paper tape gear, etc. The only fly in the ointment is that you will have to accept that you are an experimenter and you won't for the time being find much in the way of assistance from friends or magazine articles.

One other thing . . . you are going to have to start boning up on how to program uP systems. With any luck we'll start running some articles in *73* on this. You might even be the one to write 'em.

THE GREAT COMPUTER PERIL

All is not skittles and scones (whatever those things are) when you put a computer kit together. Sure, you end up with a gadget that is more fun than a barrel of monkeys (to coin a phrase . . . say, have you ever tried out a barrel of monkeys? . . . or even one monkey? My folks had a monkey when I was younger and monkey-owning has its joys and sorrows).

Once you actually get your computer working and doing things (Herculean task, some say), it is all too easy to forget some of the goals you maybe had in mind for mating it with ham gear and succumb to the fun of putting in some game programs. The next step is all but inevitable . . . bragging to your family about it. Once you've opened your mouth you've essentially lost your computer, for you will have one heck of a job getting them to allot you some time with it . . . particularly if you have any kids between seven and forty-seven hanging around.

Look at the bright side as you are putting together your second computer. You have to admit that this is the first time you've built anything which was of interest to the family . . . and that your image has subtly altered from being just a weirdo who spends a lot of time soldering useless ham gear together and then presents his back to his family while he uses it. You may even begin to detect some slight signs of respect, however camouflaged. If you hang around behind corners you may hear your kids bragging to their friends about your wizardry. Oh, they'll still put you down at every opportunity if they think you can hear, but you watch and see.

Well, the prices on computer kits are coming down, so maybe it's a good idea to shop around and get a second one. Unless you have the good fortune to work in a chip factory, you'll probably do much better to go the kit route. The kit people are buying chips in large lots and the prices are enough lower so you'll have a tough time matching them.

YES, BUT WHICH KIT?

Hopefully we will begin to get some articles discussing the pros and cons of the various computer kits available. For many applications, there is less than a significant difference between systems based on the different microprocessor chips.

MITS got in there first and this has a lot of advantages for the home builder. A great deal of competition has built up aimed at supplying boards and peripherals for the Altair system. Thus the bus structure of the Altair has become an accepted bus standard. MITS further locked much of the microprocessor industry into their bus by bringing out their new 6800 based board using the same bus and plug compatible with the Altair 8800 system.

What's a bus? It's the wiring between all of the components of the computer. Rather than having each printed circuit board of the system

Continued on page 19

by
Allan S. Joffe W3KBM
1005 Twining Road
Dresher PA 19025

What's That In Binary?

My first introduction to the world of computers left me with a reaction similar to the one I had when I saw my first schematic of a single sideband rig. The HHW* flag came up in every register.

I also tend to break out into a rash when real math comes my way, which is not the best reaction when starting to shake hands with computers.

Since others of the ham fraternity may suffer similar reactions, I have dredged the contents of my SVM** to recall a simple way of converting a base ten number into the binary numbering system or into the octal system of notation.

Interestingly enough, I was exposed to this method some thirty-five years ago, in, of all places, a Latin class. The instructor believed in making things come alive by spending some class time showing us how to multiply and divide Roman numerals. Computer stuff, by comparison, is mere child's play – and it is no wonder the barbarians wiped out Rome. It had to be painfully obvious that the Romans were so busy XXVIIing it that they had no time left to fight a mere war.

Here is a little something for my fellow non-math lovers:

$$A_{1b} + A_0 + \sum_{1}^{\infty} a^{-1}\, b^{-1} \ldots$$

It is perfectly obvious that this little bit of razzle-dazzle contains all the wisdom needed to convert a number of any base or radix to a number with any other base. For example, a decimal number to a binary number or an octal number to a decimal number.

Rather than trying to unscrew the unscrutable by translating this little gem into basic English, let me pass on a simple nuts and bolts nugget of wisdom derived from said mathematical mouthful.

Specifically, let's examine a way to convert any decimal number, such as 19758 or a number of your choice, to binary or octal. All you need is the native ability to divide by two or by eight.

Decimal to Binary Conversion

1. The digits as derived are set down right to left, the rightmost digit being the least significant bit and the leftmost digit being the most significant bit.

2. We divide the number first into odd or even by inspection. If the number is even we automatically make the least significant bit a ZERO. If the number is odd, we make the least significant bit a ONE.

3. Now we proceed to divide the number by two in a series of successive divisions. We ignore fractional remainders produced by the divisions, i.e., 19 divided by 2 would produce a real world answer of 9½, but we would ignore the remainder or fractional ½.

4. Any division that produces an EVEN number gives us a ZERO in our process of converting decimal to binary.

5. Any division that produces an ODD number gives us a ONE in our process of converting decimal to binary.

6. The process of successive divisions ends when the number is finally reduced to ONE again, ignoring any fractional remainders.

Example: Convert Decimal Number 38 to Binary

1. By inspection, since 38 is an even number the least significant bit is a ZERO.

. 0

2. 38 divided by 2 = 19. Nineteen is ODD, hence next bit is a ONE.

. . . . 1 0

3. 19 divided by 2 = 9 (ignore fraction). Nine is ODD, hence next bit is a ONE.

. . . 1 1 0

4. 9 divided by 2 = 4 (ignore fraction). Since 4 is even, next bit is a ZERO.

. . 0 1 1 0

5. 4 divided by 2 = 2. Since 2 is even, next bit is a ZERO.

. 0 0 1 1 0

6. 2 divided by 2 = 1 (divisions end). Since ONE is ODD, the next bit is a ONE.

1 0 0 1 1 0

The final binary number thus produced is decimal 38 converted to its binary form.

This painless method makes it much easier to generate decimal to binary conversions than remembering the absolute values of each binary bit, particularly for conversions where the decimal number is four or five digits long.

Decimal to octal conversion is a similar process involving successive divisions by eight. However, the signposts in the divisions are different. In the decimal to binary conversion we completely ignored the fractional parts associated with the successive divisions. In the octal conversion we use the numerator of these fractional parts to tell us what the octal bit values are to be.

As you know, in binary or base two, we only have a string of ONEs or ZEROs in the final conversion. In octal or base eight we use the numbers ZERO through SEVEN.

*How the Hell does it Work?

**Somewhat Volatile Memory (mine).

Decimal to Octal Conversion

1. As in the binary conversion, we are using a series of successive divisions; this time the constant divisor is 8. The same rule of bit value applies: Rightmost digit is the least significant bit and the leftmost digit is the most significant bit.

2. Select a number in decimal to be converted to octal and divide it by 8. This will result in a whole number or a whole number plus a fraction. If the result of the division is a whole number, the octal bit is a ZERO.

3. If the division results in a whole number plus a fraction then the octal bit is represented by the numerical value of the numerator of the fractional part.

4. The next successive divisions divide the whole number part of the previous division by 8, applying the above rule for determining the bit value of the octal number.

5. The successive divisions come to an end when you are left with a simple fraction.

Example: Convert Decimal 525 to Octal

1. 525 divided by 8 = 65-5/8. Thus our least significant bit is 5.

. . . 5

2. 65 divided by 8 = 8-1/8 thus our next bit is 1.

. . 1 5

3. 8 divided by 8 = 1-0/8. Since ONE is a whole number the next bit is a ZERO. The form 1-0/8 is shown to make it clear that the numerator is really still our guidepost even in the case of "no" fractional remainder.

. 0 1 5

4. We now divide ONE by 8 (which equals 1/8), which produces our most significant bit – and the division process ends as we are down to a simple fraction.

1 0 1 5

Thus 525 decimal has been converted to 1015, which is its octal equivalent.

These simple conversions will make life a bit more livable when you meet the computer, and you are a giant step ahead of good old Flavius Maximus, who had to MCXVII it all the way to the Circus Maximus checkout. By the time he got his change counted, the show was over! ■

I/O Editorial

from page 17

plugged into the other boards as we do with most electronic systems, all of the boards plug into a common set of wires and the switching of signals is done with programming rather than real switches. It's a lot easier that way and saves on hardware as well as cabling.

Many firms are now providing kits of parts for boards which will work with the Altair system . . . memories, interfaces, clocks, etc. If you want to use a teletype machine with your computer you have to have a circuit to take the information from the bus and send it to the teletype. It has to ignore bus data intended for going in or out of memory, in or out of tape recorders, television typewriters, etc. Your interface boards only pass that information along addressed to a specific destination . . . such as a teletype . . . and they convert the information into the form needed for the unit being served. In the case of a teletype it would convert the bus info into a 20 mA series of pulses (or 60 mA if you are using a slower and older machine). For a cassette recorder it would convert the info to a series of frequency shift tones. Okay?

In selecting the kit of your choice you have a lot of factors to consider. Since all of them are pretty good, you won't really lose by throwing a dart . . . but it is possible to get a good enough grasp on what is happening to make an educated selection.

The older kits have the advantage of being much better supported with info on problems and programs. As you get more involved with computers you will begin to understand the overwhelming importance of programming. This is a new concept to most of us and we are inclined to give it short shrift. Don't. There is an awful lot to be said for a lot of software support for a system.

On the other side of the coin, newer kits are generally able to work a lot faster and have more sophisticated abilities. Progress has been extremely rapid in the microprocessor chip design field and there are substantial differences between older chips and the new ones. A dilemma. The newer chips are better, but we are going to have to wait a while before we can get much in programming for them.

One comforting aspect to this is that the above dilemma is not really of significant importance to stop you. Practically speaking, most of the expense of a computer system is in the interfaces, memories and peripherals . . . and there is no sign that we are going to have any breakthroughs in these which will obsolete your systems. You can start out with an 8008 based system and add memory, teletype, TVT, and use the wealth of data available for this older system . . . plus all the programs . . . and then change the CPU to a 6800 system and keep everything but the microprocessor itself.

One fact you are going to have to face . . . hard fact . . . programming is not going to be very meaningful to you until you have some hardware to get your hands on and learn by doing. Oh, we can explain the fundamentals of programming in articles, and then give you lots of clever ideas for improving your programming skills, but like Morse Code, you are on your own when it comes to really learning how to program.

Once you start to get the hang of programming you'll have a ball with it. With some of the systems, you can program in just about anything that you would normally expect would require switches to accomplish . . . converting from ASCII to Morse Code . . . at whatever speed you want . . . or from Morse to ASCII . . . or to Baudot. You can route the output to your rig, a repeater, lights, a teletype, a cassette recorder, door chimes . . . anything you want. You can program music generation . . . a whole bunch of people are getting off on computer music these days. You can generate fantastic art forms by plugging into a color TV set . . . and there is a rapidly growing group of people working on computer art.

You can program your computer to compare any input against a desired data. You might tune in the Congressional Record on RTTY and set your system to check for certain key words such as Amateur Radio . . . and whenever this came up in the text your teletype would start printing.

Or perhaps you'd like to have your system switch your 2m receiver through all of the local repeaters constantly, checking for anyone sending a certain series of tones which would be your calling code. You get the picture.

CASSETTE I/O STANDARDS

One of the least expensive bulk memory systems so far available uses the garden variety cassette recorder. Those 4K and such memories in the computers are used to sort out and process data in the computer . . . the system still needs some place to store data for later use. The data may be programs to get the computer to play certain games, it might be names and addresses for a mailing list, book or record lists for an index, etc.

Eventually we will have other bulk memory systems which are reasonably low in cost. At present a floppy disk memory runs around $2000, which puts it rather in the expensive area for most home systems. A disk can hold around 250,000 bytes of memory and make it available within a few microseconds . . . good business if you are going to sort data or want quick access to random things.

Regular computer tape drives are not cheap . . . yet. They use half inch tape (usually), and cost $2000 or so, too. They can hold a lot more memory, but the time required for searching through the tape is much longer. There are some quarter inch tape cartridge systems in evolution which may be low in cost for home computers, but nothing is yet ready.

All of which brings us back to cassettes.

When I began to get an understanding of what was involved I looked to see what standards were being used for putting information on cassettes. I found, much to my horror, that there seemed to be as many standards as there were firms coming into the field. This would never do! We needed one standard for cassettes so that any program or memory cassette would work in any system regardless of manufacturer.

Since there was no one else to tackle the problem, I figured it might as well be me. I sent letters out to everyone in the industry that I knew about suggesting a meeting of everyone involved on neutral ground. I picked Kansas City because that was equally far away for everyone. I scheduled the meeting for November 7-8th.

The meeting came off right on schedule and just about everyone from the microprocessor industry turned out for it. At first the meeting got off to a very slow start and it looked as if no agreements would be possible, but (as I suspected) the spirit of cooperation won out and everyone finally got together and hammered out a standard which was agreeable to all.

The standard calls for two tones to be used, a mark tone at 2400 Hz eight cycles long for logic one and a space tone 1200 Hz and four cycles long for logic zero. A recorded character will consist of a space (start) bit, eight data bits, and two or more mark (stop) bits. The eight data bits are organized with the least significant bit first and ending with a parity bit, if used. All unused bits will be mark tones . . . e.g., the three unused bits when you are sending five level data. Mark tone will also be used in between characters as fill.

This system will give you about 47K bytes of data on a 30 minute cassette (one side of a C-60). If you use all four cassette tracks that would add up to almost 200,000 bytes. To put that in terms of an application, figure that you are putting names and addresses on the cassette. If you take 80 bytes for each name and address (which is what we use for the *73 Magazine* subscription records), this would permit you to have over 2000 names and addresses on a single cassette. That's a hefty amount of storage. If you are going to use an inexpensive recorder you would probably only put half that much.

Frank Kelly W2IAT
49 Knollwood Avenue
Huntington NY 11743

Say, OM, Are You a Computer?

As IC construction projects become more and more complicated, it becomes clear that the logical endpoint is a gadget that handles the whole QSO – sends out a CQ, answers the call, decides on a signal report, describes the rig, asks for (and promises) a QSL, and taps out a sincere "73" at the end. A recent *QST* project, the Contester, was in fact a fairly sophisticated unit for handling all of the bookkeeping in contest operation. In that article and others, the prediction of the computer-controlled ham station is made.

I've just seen a piece of that station in operation. Robert Snider, a 14 year old high school student in Cold Spring Harbor, New York, has configured a PDP-8 minicomputer to send and receive the International Morse Code. The interesting part of this is that it was not a construction project; Bob never heated up a soldering iron or bored a hole in a panel. Instead, as an exercise in *programming* for a math project, Bob essentially "taught" the code to the computer and devised an ingenious way for it to send and receive.

As computers go, the PDP-8 is a pea-brain. In reality, it owes its fame and success to the fact that it does less and does it more slowly than the computers that preceded it on the market. By being a not-too-bright midget, the PDP-8 started what some have termed "the minicomputer revolution".

Almost from the first machine that deserved to be called a computer – a room full of vacuum tubes at the University of Pennsylvania – the evolution of computers has been that today's machine is faster, does more, and has a larger memory than yesterday's. This progression of smarter and smarter machines – with higher and higher price tags – was in full flower when the designers of the PDP-8 realized that there were many jobs that could be handled by a computer of modest accomplishments. These jobs at that time were being done by laboriously designed collections of special circuitry because a simple (therefore cheap) computer just didn't exist. So the bright guys at Digital Equipment Company took a fair-sized step backward and introduced the PDP-8 to what turned out to be a waiting world. The PDP-8 is slow, it has a limited memory, and as a mathematician it is pretty poor – it can only add. If you want subtraction you have to add a negative number. Multiplication and division are implemented by software subroutines. However, the PDP-8 was, for its time, small in size and dirt cheap. It became possible to think of the computer as a component, the controller of a larger system such as an automated drafting table, or the overseer of a complicated process like the management of a portion of a pipeline system.

Since 1958 when the PDP-8 was introduced, computers have, been built into hundreds of systems, and the PDP-8's successors and imitators have proliferated until there are now scores of minicomputers on the market. The PDP-8 (shown with Bob Snider in the photograph), has been slimmed down considerably. Originally it was an ungainly 3 foot high unit with its PC boards showing. One manufacturer is even offering the "Naked Mini" – essentially a PDP-8 minus power supply, switches and cabinet – on a single giant PC board. Minicomputers can now be had, with performance equal to the PDP-8, for about $1500. Considering what the plutocrats in ham radio spend on their rigs, this is the right ballpark for inclusion in that "super station".

The job of programming a computer to handle the code was basically a simple one, according to Bob Snider. His ingenuity lies in the way he developed for the machine to send the code, without any electrical connections. For those of you who have never met a minicomputer, it's basically a RTTY op. You type to it; it types back to you. Since Bob's goal was audible code, this wasn't satisfactory. And since the computer is owned by Cold Spring Harbor High School, getting inside to make a wire connection to a keyer or code oscillator was forbidden. But Bob discovered that if he put the computer into a programming loop (in which it endlessly repeats the same operation), the machine generated enough stray rf to be heard on a nearby broadcast receiver. *Voila*, the wireless transducer!

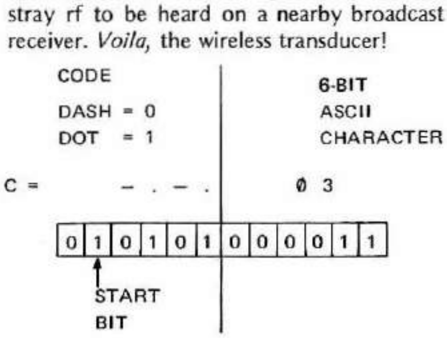


Fig. 1.

The program thus switches in and out of a loop for appropriate lengths of time to generate dots and dashes corresponding to the text that has been typed into it. Speed is no problem; it asks you what you'd like. It's equally comfortable with one or 99 wpm. In fact, it could go faster but for one thing which Bob considers a minor goof – he made the slot where you insert the desired speed only two digits long.

How is it done? Bob explains that the PDP-8 has a twelve bit word – that is, the basic building block of any program is a series of twelve binary digits. This also corresponds to one memory cell. The keyboard outputs 6-bit ASCII characters and the task of the program is to match these up with dots and dashes. Now, matching is something even a simple computer does well. So the six-bit ASCII character is inserted in the second half of the word, and the correct dots and dashes in the first half. How it works for the letter C is shown in Fig. 1.

The entire alphabet, plus the numbers, is stored in a table of these words in memory. When you hit a letter on the keyboard, the keyboard output (ASCII) is compared to the right half of the words in the table. When a match is obtained, the left half (code) is placed in a register and shifted left to the start bit, always a 1. This bit is ignored, but the next bit causes the loop to run to produce tone – for three units of time if 0 (dash) or one unit of time if 1 (dot). The units of time have been computed by another section of the program to be appropriate for the speed you asked for. Succeeding bits produce dots or dashes until the end of the six-bit half word (or "byte"). All characters are right justified, so the byte end serves as a stop signal.

This is the "SEND" mode of the program in which the computer sends to you. The "RECEIVE" mode has the human operator sending and the computer translating to typed characters on the teletype. No key could be connected by wires, remember, but Bob recollected that the "BREAK" key (used generally to interrupt a program in process) is the one key on the keyboard that gives a simple contact closure; no ASCII code is generated. So this key is used as a telegraph key.

The break key is not ideal for this purpose. I couldn't send the computer anything it liked. It kept typing out ERROR in response to my sending. It turns out to be rather finicky as to the rhythm and spacing it will accept. It reads Bob Snider like a champ, though.

I should add at this point that the program wasn't designed to duplicate a code typer or code reader. Instead, it was conceived as a teaching machine to teach the code to complete neophytes. For this reason, it has other modes than "SEND" and "RECEIVE". In the "LOAD" mode, the user types a complete message into memory, which on the SEND command the machine transmits to the radio. Proper character and word spacing for the selected speed are automatic.

In the "LOOK-UP" mode, the user types any letter or figure, followed by an equal sign, and the computer prints the code character using periods and hyphens for dots and dashes. There's even a "TUNE" mode, in which the computer "holds its key down" so that you can adjust the radio's tuning and volume for best reception.

This program won the top award for Bob Snider in the Long Island Mathematics Fair. He submitted a paper describing the program, and then, on March 22, 1974, gave the paper orally and demonstrated the technique for a panel of judges. In competition with the work of the brightest math students from schools all over Long Island, Bob's project won. In talking to him and seeing the demonstration, I got the distinct feeling that he considers this program kind of old hat by now. It turns out Bob Snider developed this when he was in the 8th grade. His advisor also told me that Bob has won the Long Island Mathematics Fair three years in a row, a feat about as common as a team winning the World Series three years in a row.

Bob's into other things now – reprogramming the computer so it can be time-shared among several users, and possibly handle the Cold Spring Harbor High's attendance and scheduling records as a side job.

He admitted he's too busy with these things (and a new sailboat) to think about getting a ham ticket. So if you do contact a computer one of these days, it probably won't be Bob Snider behind the scenes.

It also may not be in the CW portion of the band. While I was at the school, another teen-aged programmer, Craig Hansen, demonstrated his latest program for me. It seems Craig's program makes the PDP-8 literally *talk* – in a pleasant baritone voice that's completely synthesized from binary numbers. Not perfectly though. It speaks a little too slowly for contest work . . .

. . . **W2IAT**

I/O EDITORIAL

COMPUTERS, SIMPLIFIED

Computers are complicated, but the basic parts of them are not difficult to get to know, and from there on it is a matter of a whole lot of the same thing over and over, so the complication is one of quantity.

For many ham applications of computers you can think of them as being an almost endless switchboard which can be operated by means of a keyboard rather than individual switches or the soldering of wires. Take a repeater, for instance. Right now repeater nuts have to wire in a time delay for the squelch tale, another for the three minute time out, an ID generator, another timer to play the ID at desired times, an autopatch decoder, and so forth. With a microcomputer, said nut can do all of that with a program. And if he wants to change the length of time for the squelch tale he can go on the air. access the repeater control computer and change it. Ditto the three minute timer . . . perhaps he'd like to shorten the drop out time to two minutes during drive time . . . no strain. The uP will turn on the cassette recorder during autopatches, etc. It is an almost infinitely flexible way of simulating a lot of electronic circuits. Now do you understand what all the fuss is about?

Amateurs have, for the most part, a decided advantage over most of the people who are already into computers. Most of these people are into the programming end of things and it just happens that programming is a lot easier to learn than electronics. We can learn programming quickly, but they will have a long hard row to hoe trying to catch up with us on the digital electronics end.

Well, perhaps you have no great interest in founding a multi-million dollar firm in microcomputers . . . no interest in grabbing a piece of the big computer pie in the sky which is coming. The estimated ham purchases of computers and accessories for this year is only about $50 million . . . let's see now, if we only get 2% of that market that is $1 million in sales. Hmmm. And once a few more people get hold of the Digital Equipment book on computer games the computer hobby market could explode. Fortunately DEC has managed to keep this book a tightly held secret . . . otherwise all hell might break loose. 101 utterly fascinating computer games and the programs to get them running all in one $7.50 book . . . fantastic.

With home use, hobby use, ham use, and business use, computers will be everywhere in a few years. Offices will use the same computer for bookkeeping, invoicing, writing letters (watch out IBM, your $850 Selectric typewriters are about to be made obsolete), inventory, addressing, payroll, indexing, and anything else that now uses paper, pencil, typewriter, etc. Your friendly local computer store will sell, program and service said ubiquitous computers. It is not often that we can see a multi-billion dollar market about to open up and decide whether we want to grab the shirttails or not.

CHU DIGITAL TIME

CHU, the Canadian Observatory station, has added a digital time function to its transmissions. This time tick uses the standard modem tones of 2025 and 2225 Hz and is sent during the 30-39th seconds of each minute. Who will be the first to come up with an article on using this time standard for a ham shack clock or computer application?

The time code is a modified ASCII at 300 baud. During the first 500 ms of each of the 30th to 39th seconds, CHU sends first 10 cycles of 1 kHz (bringing us up to 0.01 seconds . . . 10 ms), then 125 ms of 2225 Hz. This is followed by two bursts of the time code (182.5 ms each). If the two time bursts are not identical the pulses should be ignored.

The time bursts are made up of five 8-bit characters, each containing two decimal numbers (BCD), for a total of ten digits in the pulse. The first digit must be a six as this will prevent inversion of the code. The next three indicate the day of the year, then two for the hour of the day, two for the minute, and two for the second. 2025 Hz is level 1 and 2225 Hz is level 0.

Gearing up to use these digital time clicks is a fine project for you digital home brewers.

Continued on page 32

Jeffrey Mogul WA1PAZ
218 Franklin St.
Newton MA 02158

Computers are for Hams!

During the last few years, there has been a dramatic growth in the availability of digital computers to the average person – or ham. Almost any college or university now provides computer access to its students, and many high schools now have some computer facilities. This is due, in part, to the equally dramatic drop in prices for small computers, also known as minicomputers, which range in cost from about $1500 to as much as several hundred thousand dollars. Another factor is the increase in large time-sharing systems, which make computer time available to anyone with a telephone and a paid-up account. If you can't see spending even a few thousand dollars for a mini, you can always buy a microcomputer. MITS sells its Altair model for $439, and complete kits using the 8008 chip have been advertised for under $400. If you have some of the makings, and like to experiment, you can pick up so-called microprocessor chips for under $30.

But computers are useless unless they are programmed. A computer program is like an instruction manual that the computer can use to solve the problem which you want it to solve. Computers are dumb; they can't do anything without explicit instructions. If programming scares you, there are many programs that have already been written; all that you have to do is to feed them in and start them up. But it is cheaper, and much more fun, if you write your own programs. Besides, not every problem which you may want to solve has already been written as a program.

Writing programs requires two things of the programmer. The first is an algorithm – which is simply a method of solving the problem. But as was mentioned before, computers need very explicit instructions to do anything. Each computer understands one, and usually only one, language. This language is nothing like a human language. In fact, to a computer, a typical statement in computer language might look like this: 1 1 1 ØØØØØØ 1 ØØØØØ 1 1 Ø. That particular statement would tell a PDP-9 computer to print one letter on the teletype connected to it. If one had to write all programs like that, we would still be using slide rules to send men to the moon – if we ever wanted to try. But fortunately, someone decided to take it upon himself to write, just once, a program in this kind of form which would translate a more usable language into 1s and Øs. Once this was done, you never had to write a program in 1s and Øs; instead, you could use little combinations of three or four letters called mnemonic op-codes, usually called simply op-codes. (Try to pronounce mnemonic!) For example, the sequence of 18 1s and Øs above could be written like this: TLS. The program which does all of this is called an assembler, and you usually can get one from the manufacturer of the computer. Unfortunately, you must get a different one for every computer, and usually you must learn a new set of op-codes for every computer.

Anyway, now that we have a language that both we and the computer understand, we can take our algorithm and put it into our computer's language, feed it in, and the computer should solve the problem. Much of the commercial and scientific programming is done this way. But to write a program in op-codes is very tedious, because most op-codes express very simple ideas. Therefore, some geniuses wrote programs which would translate a high level language into op-codes. A high level language is one which even a junior high school student can use, because it is very simple to understand. Some of the widely used high level languages are FORTRAN, BASIC, ALGOL, COBOL, APL, FOCAL, and PL/1, to name a few. In BASIC, the statement: PRINT "HAM RADIO IS FUN" would cause the teletype to print out HAM RADIO IS FUN. The same procedure would require 42 op-code instructions, or 756 1s and Øs if done on a PDP-9 computer. The program which translates high level language to op-code language is usually called a compiler.

One of the most common ways of using a computer is to use a time-sharing system. This is when anywhere from 2 to 2,000 (or even more) people are connected simultaneously to the same computer. Each person uses the computer as though he were the only one using it, and the computer is fast enough to give each user a little time every few milliseconds, so that it seems as though it is devoting itself completely to each user. This is like a movie; the pictures aren't really moving, just presented fast enough so that we think they are. If you use a time-sharing system, you will probably use a high level language, since most time-sharing systems do not allow use of op-code languages. The two simplest languages to use, and therefore most common on large time-sharing systems, are BASIC and FORTRAN. FORTRAN, which stands for FORmula TRANslation, is not as well suited to time-sharing as is BASIC, and is somewhat confusing to the neophyte programmer.

BASIC stands for Beginner's All-purpose Symbolic Instruction Code, and was the first time-sharing language. It was developed at Dartmouth College, and the primary goal of the developers was to give the programmer as easy a time as possible. Many other, more

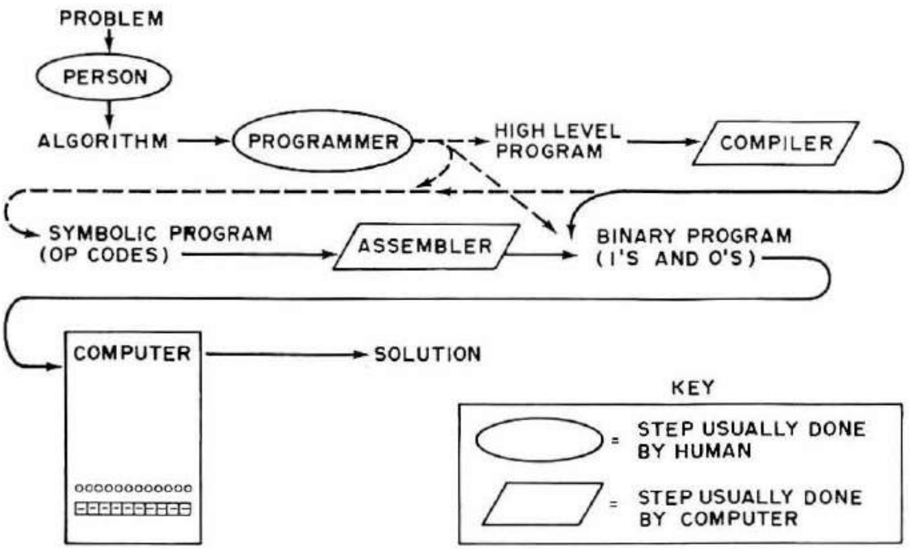

PROBLEM
PERSON
ALGORITHM
PROGRAMMER
HIGH LEVEL PROGRAM
COMPILER
SYMBOLIC PROGRAM (OP CODES)
ASSEMBLER
BINARY PROGRAM (1'S AND 0'S)
COMPUTER
SOLUTION
KEY
= STEP USUALLY DONE BY HUMAN
= STEP USUALLY DONE BY COMPUTER

Fig. 1. Procedure for solving a problem by computer.

advanced, versions of BASIC have since been written, but they share the feature that a program can be written in the simplest version and be usable with any of the advanced ones. As one learns more about programming, more and more features can be added to one's computer "vocabulary." BASIC compilers are available from almost any minicomputer manufacturer, ranging from the Digital Equipment Corporation (DEC) PDP-8, at about $2000, to the Wang Laboratories $7,400 System 2200, which has the language wired into the computer itself, to the $80,000+ RSTS/E system used with DEC's PDP-11 series. (DEC, incidentally, is the biggest computer success story since IBM – there are more of their little PDP-8s floating around than any other computer.)

So what use is the computer to the average ham, you ask? Almost any area of ham radio activity can be helped by the computer. For example, my radio club now produces mailing labels for our newsletter on the computer. We put all of our membership information onto punched cards, feed them into the computer, and out comes a gummed address label for each member. We used to sit up for several nights, copying names by hand from a list to the envelopes. With the same stack of cards, we can produce the club roster in several minutes, all set to go on mimeograph masters.

Even if you don't belong to a radio club, you can still use the computer. OSCAR users can get lists of OSCAR orbits for any area and for several years at a shot. DXers can make personalized tables of beam headings for their most important targets. Computers can print QSL cards, keep and check contest logs, predict spurious outputs for FM-27s, predict receiver birdies[1], coordinate DX hunting[2], design and test new circuits, graph antenna patterns and produce the Callbook. I have heard about repeaters that are completely controlled by a microprocessor. The computer can solve virtually any problem which requires the manipulation of numbers or data. I doubt if anybody has tried, but it should be possible to have computer chess tournaments over RTTY or slow scan TV. Some chess-playing computers, like Chess 4.0 of Northwestern University, have achieved high intermediate ratings.

It makes little sense for every ham to write his own programs for every problem he wants to solve ... in fact, there are many organizations which already exist for the sole purpose of exchanging programs between programmers. For example, Digital Equipment Computer Users Society (DECUS), sponsored by DEC, provides such a service. Catalogs of programs are sent free to members, and paper tapes of programs are available for little cost. Members contribute programs and articles to the society magazine, DECUSCOPE. DECUS holds conventions several times each year. Within DECUS are many Special Interest Groups (SIGs), which appeal to smaller segments of the programming fraternity. There is room for a Radio Users Group (RUG?). DECUS will provide, for free, certain services for any SIG, such as mailing and printing of newsletters. Perhaps some organizations, like ARRL or IARU, could get together with DECUS and support a RUG. Other computer manufacturers, MITS for example, already provide similar services for their customers, but there are considerable advantages in having one organization handle all amateur programs. Although most programs are only usable on one computer, the algorithms behind them can be translated into any language. Algorithms are sometimes expressed in "flow charts," and these flow charts can be exchanged as easily as programs. I think that, in the future, articles in ham magazines on computer programs will be as common as construction articles, and programs will be distributed in the same way that printed circuit templates are.

I hope that this article has given you some idea of what the computer's place in ham radio is, and perhaps removed some of the mystery which shrouds the computer and its uses. The computer has been described as the most important technological innovation of the twentieth century, and although we all know that radio is more important, we can certainly give the computer a good shot at second place. We have nothing to lose but our slide rules!

...WA1PAZ

References

1. Gadwa, "Computerized Search For Receiver Birdies," *QST*, February, 1974.
2. Cone, "Computerized DX Chasing Made Easy," *73*, December, 1974.
3. "Introduction to Programming," Digital Equipment Corporation, 1970.
4. John G. Kemeny, "Man and the Computer," New York, Scribners, 1972.
5. Norbert Wiener, "Cybernetics," The MIT Press, Cambridge, Mass., 1961.
6. "Digital Computer Basics," Dover Publications, New York, 1968.
7. "BASIC-PLUS Language Manual," Digital Equipment Corporation.

Larry Kahaner WB2NEL
4259 Bedford Avenue
Brooklyn NY 11229

How Gates Work

A TTL PRIMER

The writing technology of integrated circuits has not kept pace with the design and application technology, as evidenced by the hunger of 73 readers for very basic articles.

It is relatively easy to find circuit designs using ICs and easier still to mimic the projects to a fruitful end although the basic understanding is still lacking.

When writing a book on tube electronics it is simple to know where to begin. It has been standard practice to start with the concept of atomic structure and electron flow off a hot filament. With transistor texts, we begin with atomic structure of semiconductor material and explain the movement of "holes". Digital IC theory does not require us to go microscopic, but it does call on us to think in new ways about electricity, new and unique ways. It is the purpose of this article to start the amateur on this new way of thinking; logically!

Integrated circuits are a complex interconnection of circuit elements within one continuous structure. An IC "chip" may contain the equivalent of fifty transistors and resistors in one package. The TTL ICs discussed in this article are of a digital nature (to be defined later) and so called because each chip can have its function imitated by a myriad of transistor circuitry. In other words, we can do everything a TTL IC can do with transistor-transistor logic circuits, but they will be bulkier, costlier and less efficient.

For the purposes of this discussion let's think of electricity as existing on different levels of value instead of different voltages. For example: We can assign a voltage of 5 volts the value 0, and a voltage of 10, the value of 1. It is customary to give the higher voltage the value of 1 and the lower, 0. If we have a square wave function we can say that the wave fluctuates from a level of 0 to 1 and back. Since the change is almost instantaneous in a wave of this form, the voltage can be thought of as having only two states, 0 and 1. The idea of discrete steps or states is known as digital. The states can be used for yes-no, on-off, or any meaning you may need.

LOGIC GATES: EASY STUFF

The AND Gate

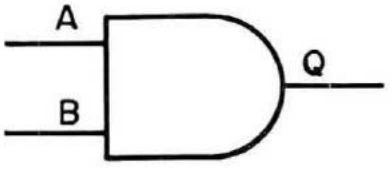


Fig. 1.

Fig. 1 is the symbol for the AND gate. There are two inputs, A and B, and one output, Q. There can be more than two inputs. The concept of AND can be put into everyday terms.

Suppose I said, "I can drive my car only if I have my KEYS *and* GAS." I cannot drive my car if I have only my KEYS or only GAS. I must have both simultaneously. Table 1 is a "Truth Table".

Keys?	Gas?	Drive the Car?
NO	NO	NO
NO	YES	NO
YES	NO	NO
YES	YES	YES

Table 1.

If we let the symbol 0 mean NO and 1 mean YES, we have the following truth table for the AND gate:

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Table 2. AND gate truth table.

This can be expanded to three inputs by saying that to drive my car I must have KEYS *and* GAS *and* EYEGLASSES. To have two out of three is not sufficient; we must have all three together to obtain an output (drive car).

The NAND Gate

The NAND gate is drawn as shown in Fig. 2.

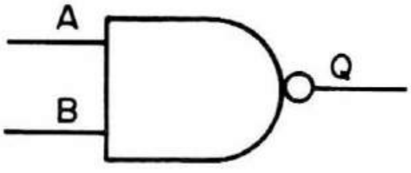


Fig. 2.

The little pimple on the end of the AND gate means "negation". It says, take the AND gate and make a truth table. But, take every answer and negate it. If it comes out 1, make it 0; if it comes out 0, make it 1. That's all.

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

Table 3. NAND gate truth table.

The OR Gate

The OR gate is not used as much as the AND and NAND, but it is important nonetheless. Here is how it looks:

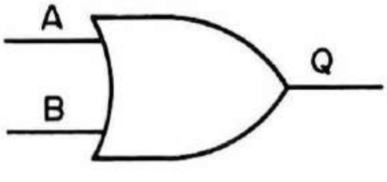


Fig. 3.

An example of its function is to say, "I will have a pleasant day if I GET ON THE AIR or GO MOTORCYCLE RIDING."

Get on Air?	Motorcycle Riding?	Pleasant Day?
NO	NO	NO
NO	YES	YES
YES	NO	YES
YES	YES	YES

Table 4.

As you can see it is not necessary to do both, but at least one. Of course if I do both, I will have a pleasant day, but either is sufficient. The truth table is shown in Table 5.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Table 5. OR gate truth table.

The NOR Gate

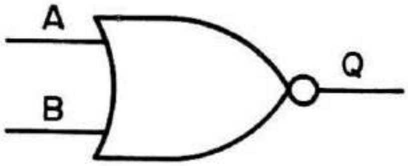


Fig. 4.

As the NAND negates the AND, the NOR is the negation of the OR. Whatever the output of the OR gate, negate it. Table 6 shows a NOR truth table.

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Table 6. NOR gate truth table.

The last simple gate is the NOT gate. Also known as the INVERTER. Its symbol is:

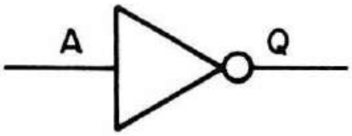


Fig. 5.

Its function is the simplest one of all. It inverts or negates the input. It can have only one input and only one output.

A	Q
0	1
1	0

Table 7. NOT gate truth table.

There are two more gates worth mentioning but they are not considered simple gates like the ones above.

They are the EXCLUSIVE OR gate and the AND OR INVERT gate. The EXCLUSIVE OR gate has a truth table similar to the OR, except that it has no A=1, B=1, Q=1 state. It is exclusive of this logic sequence.

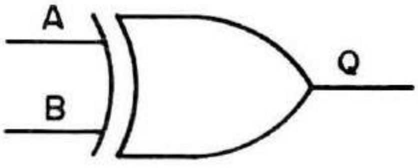


Fig. 6. EXCLUSIVE OR gate.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Table 8. EXCLUSIVE OR gate truth table.

The AND OR INVERT gate is a combination of AND and NOR gates and looks like this:

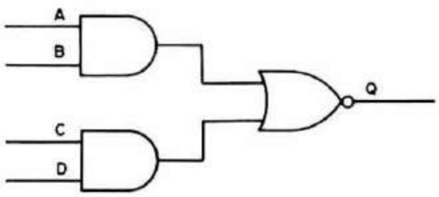


Fig. 7.

To construct a truth table, take each gate individually and follow through to the output. It may be helpful to call the inputs of the NOR gate E and F, and give them each a column until you can do it in your head.

A	B	C	D	Q
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Table 9. Partial truth table for AND OR INVERT gate.

CLOCK INPUTS: A LITTLE TOUGHER

The square wave previously discussed is very often seen in digital circuitry. It is used in clocks and all types of counters.

If we take an AND gate and impose the square wave (also called a clock input) on input A, we can think of it as going from logic state 0 to 1 to 0 to 1 etc . . . If we then keep input B fixed at logic state 0, the output at Q will be 0. Again look at Table 2. But if we make B=1, then Q will be 0 when A is 0, and 1 when A is 1. We have an unchanged input from Q.

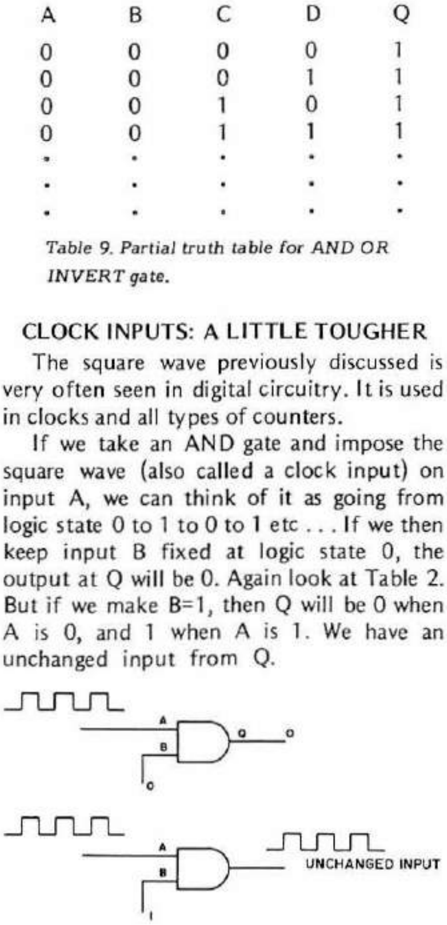


Fig. 8.

The "trick" to doing these is to think of the input as a movement of discrete points and to do the logic for these points. Here is the NAND gate again:

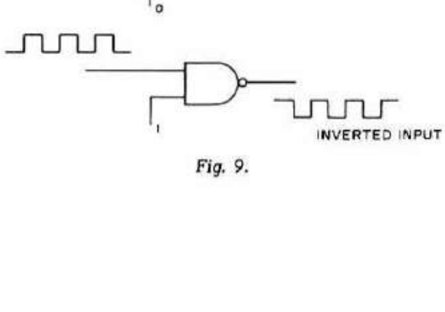


Fig. 9.

The OR and NOR gates follow in like manner:

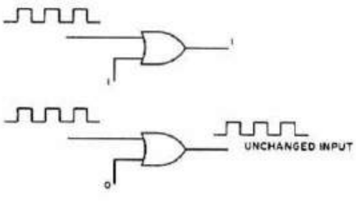


Fig. 10. OR gate.

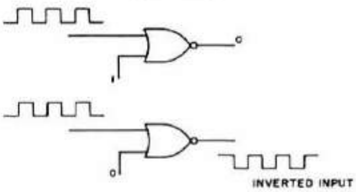


Fig. 11. NOR gate.

GATE TO IC: THE MISSING LINK

If we could look inside the 7400 chip, we would see this:

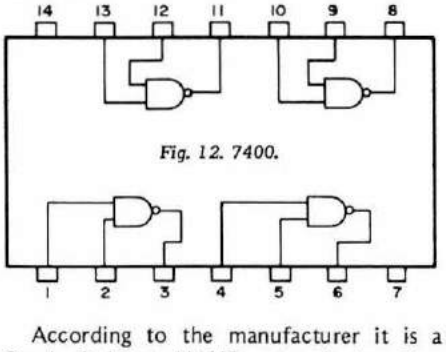


Fig. 12. 7400.

According to the manufacturer it is a Quad, 2 input NAND gate. It has four (quad) gates of which each has 2 inputs, and, they are NAND gates. Pin 7 is ground and 14 is the voltage supply. We can use all or any of the gates. Gates can be connected together if needed for a certain function.

Not all chips can be shown in an "X-rayed" view because some are so complex that drawing is impossible. Such is the case in the 7490 chip which is shown like this:

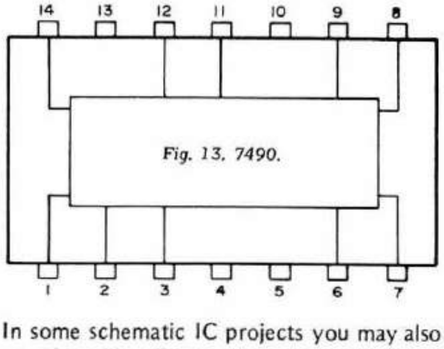


Fig. 13. 7490.

In some schematic IC projects you may also see the chip divided into gates separated from each other. It is easier to break it up than draw the chip in the schematic in one piece. It is also easier to trace the logic with it broken down to gate functions.

Well, believe it or not, that is all there is to digital TTL ICs. That is, the 7400 series. And there are quite a lot of them.

. . . WB2NEL

William E. Browning WB5IRY
516 N. 95th E. Ave.
Tulsa OK 74115

The Best Logic Yet

When I first got into amateur radio just about three years ago, I had a little understanding of the way tubes worked. I decided that if I built anything for the shack it would be with tubes – and that I would just buy anything that had to use that mysterious solid state.

This decision did not last for long, however. I just don't like having to buy everything, since I'm really a builder at heart. So all that was left for me to do was start learning about solid state. I started building some projects from the pages of "73" and some other magazines – some worked and some didn't. When they didn't, I was usually lost as to why.

Then I found that the projects that used what was called TTL logic usually worked the first time and that when they didn't I could usually find out why. Over a period of a few months I had formulated a list of fundamentals that almost assured that the projects would work.

Presented here are some of those fundamentals from my notes. If you use them as a guide line I know you can do as well or probably better at TTL projects than I have.

FIRST: The power supply for TTL is one critical spot. The absolute maximum voltage for most of the TTL ICs is 7 volts, with 5 volts being normal. *Stay within 4.5 and 5.5 volts.*

SECOND: Find the output pins, and *do not connect two outputs together.*

THIRD: If you use the 5 volts from the power supply for an input signal on an input pin or for any reason put the 5 volts from the power supply to an output pin without a load, *connect a resistor in series* (any size from 100 Ohms to 10k Ohms will work in most cases).

If you follow these first three design rules you will find that it is almost impossible to damage the ICs and you can now experiment with them all that you want to.

Now that we are not going to send the IC up in smoke, let's see what we will find inside some of them and how we can put them to work for us.

The first one to look at, which is the simplest of all, is the buffer. With the buffer the output is the same as the input. It is used to isolate the input circuit from the output and also to drive more circuits than your input signal may be able to. The logic symbol for a buffer is shown in Fig. 1.

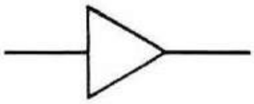


Fig. 1.

With the buffer, if we put "0" volts or ground on the input, the output will be "0" volts. If we put 5 volts (often referred to as logic 1) on the input, the output will be 5 volts (logic 1). See Fig. 2.

Fig. 2.

You will notice that the input is a square type wave; this is the type of input that digital circuits need to work properly. There are input circuits that will let us use sine wave inputs and some special ICs which are designed for a sine wave input, but for now we will stay with the square type input.

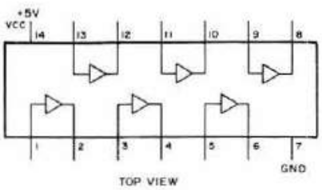


Fig. 3.

One digital IC which uses the buffer is the 7407 hex buffers/drivers. This chip has six buffers on the one IC. Each buffer can be used separately. (The layout of the chip is shown in Fig. 3.)

Note that +5 volts goes to pin 14 and that ground is on pin 7. This supplies power for operation of all six buffers. With an input on pin 1 you get an output on pin 2, while an input on pin 3 will give you an output on pin 4, input on 5 for output on 6, input 9 for output on 8, etc.

One thing that works out nice for testing is that if an input is not connected to anything (is floating), the IC sees it as logic 1 (5 volts) and the output goes to 5 volts. If the input is grounded (logic 0) the output goes to 0 volts. You can test a chip by watching the output on a voltmeter as you ground and unground an input.

FOURTH: *Consider a "Floating" input as logic 1.*

If an input is to be at 0 volts make sure that it is connected to ground.

The next "gate" to look at and experiment with a little is the inverter. It is almost the same as the buffer, with one exception: its output is always opposite from the input. The logic symbol for an inverter is shown in Fig. 4. (The only difference between the symbol of an inverter and the symbol for a buffer is the small circle at the output.)

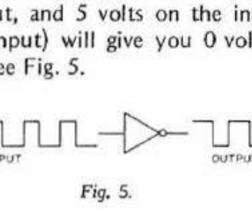


Fig. 4.

With the inverter, 0 volts or ground on the input will give you 5 volts (logic 1) on the output, and 5 volts on the input (or a floating input) will give you 0 volts on the output. See Fig. 5.

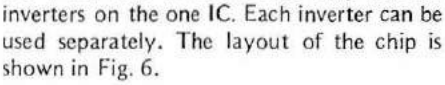


Fig. 5.

One digital IC which uses the inverter is the 7404 hex inverter. This chip has six inverters on the one IC. Each inverter can be used separately. The layout of the chip is shown in Fig. 6.

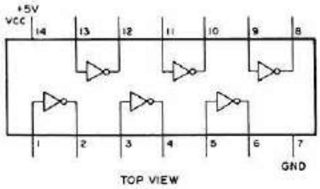


Fig. 6.

Note that on the 7404, +5 volts is on pin 14 and that ground goes to pin 7. This supplies the power for operation of all inverters. With an input on pin 1 you get your output on pin 2, while an input on pin 3 will give you an output on pin 4, etc.

It is possible to use an inverter as a buffer by putting two gates in series. As an example using the 7404, you could connect pins 2 and 3 together, place your input on pin 1 and take the output from pin 4. The output now will be the same as the input just like with the buffer. See Fig. 7.

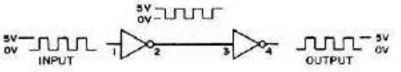


Fig. 7.

One other thing that you may want to keep in mind about the inverter is its other name: the "NOT" gate. This comes from the fact that its output is NOT the same as its input. But whether you call it an inverter or you call it a "NOT" gate, it is the same thing with the same symbol.

Another gate that you will find in the logic family is the "AND" gate. It is similar to the buffer, the only difference being that it has more than one input. The logic symbol for an "AND" gate is shown in Fig. 8.

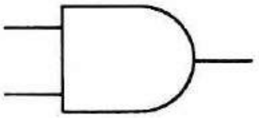


Fig. 8.

With the "AND" gate, both inputs must be at logic 1 (5 volts) before the output will be logic 1. This is where the gate gets its name: input 1 AND input 2 must both be 1 (5 volts) to get a 1 for an output. See Fig. 9.

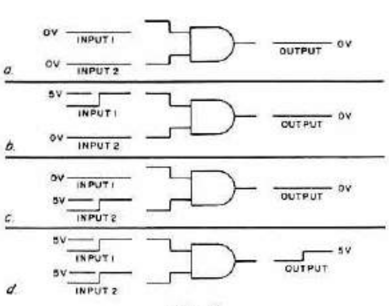


Fig. 9.

The "AND" gate can also be used as a buffer and there are two ways to do it. One way is to put a constant 5 volts on one of the inputs and put your input signal on the other. (If you use this method use a resistor in series with the 5 volt power supply.) The other way is to connect the two inputs together and put your input signal into both at the same time. See Fig. 10.

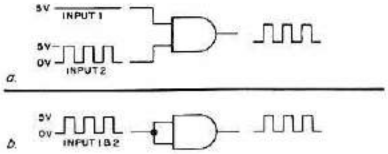


Fig. 10.

One digital IC which uses the "AND" gate is the 7408 quad 2-input AND gate. This chip has 4 "AND" gates on one IC. Each gate can be used separately; the layout of the chip is shown in Fig. 11.

As with the other chips, the +5 volts is put on pin 14 and the ground is put to pin 7. Input 1 and input 2 go to output 3. Input 4 and input 5 go to output 6, etc.

In some circuits it is desirable to have the two inputs with the AND function, plus the signal inversion of the NOT gate (inverter). This is now easy to do by using the two gates together. See Fig. 12.

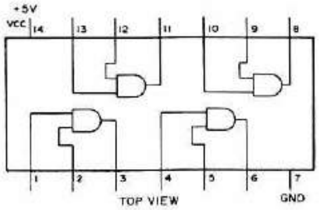


Fig. 11.

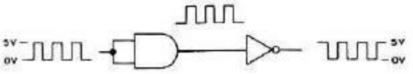


Fig. 12.

The use of the "NOT" and the "AND" gate together is a very common combination, so common in fact that it is considered as another gate called a "NOT-AND" gate or even more frequently a "NAND" gate. So remember, when you see "NAND", that it is just an "AND" gate followed by a "NOT" gate. These two gates are often combined on one chip as one gate and the symbol for the combination, the "NAND" gate, is shown in Fig. 13.

You will notice that the only difference in the symbol of the "NAND" and the symbol of the "AND is the "o" at the output of the gate. In the use of logic symbols the "o" will indicate that the signal is inverted.

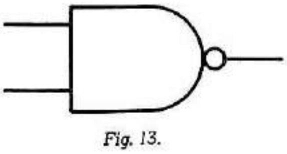


Fig. 13.

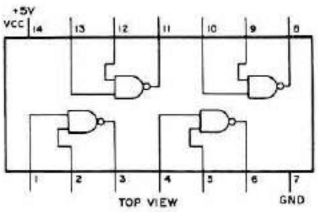
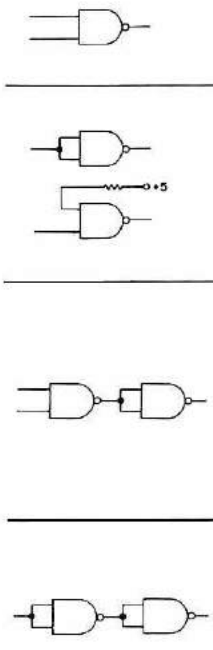


Fig. 14.

One digital IC which uses the "NAND" gate is the 7400 quad 2-input NAND gate. This chip has 4 "NAND" gates on the one IC. Each gate can be used separately; the layout of the chip is shown in Fig. 14.

As can be seen, +5 volts goes to pin 14 and ground goes to pin 7. Inputs 1 & 2 go to output 3, inputs 4 & 5 go to output 6, etc. The 7400 is one of the most widely used ICs of the TTL logic family, since it can be used as a buffer, as an inverter or "NOT" gate, as an "AND" gate, and as a "NAND" gate. Whatever the operation you have in mind, the 7400 can be made to work. Not bad for an IC that can be picked up for under a dollar. Fig. 15 shows how to connect it to work as the different gates.



"NAND" — Use each gate as is.

"Inverter" or "NOT" — Connect the two inputs together, or connect one input to +5 volts and use the other input for your signal.

"AND" — Feed the output of one "NAND" gate into a second gate that is connected as a "NOT" gate, and take your output from the second gate.

"Buffer" — Feed the output of one gate connected as a "NOT" into a second gate connected as a "NOT", and take your output from the second gate.

Fig. 15.

Another type of gate that you may come across is what is known as an "OR" gate. It is similar to the "AND" gate with one exception: With the "AND" gate you had to have 5 volts (logic 1) on both inputs 1 *and* 2 to get logic 1 (5 volts) for an output. With the "OR" gate, a 5 volt input on input 1 *or*

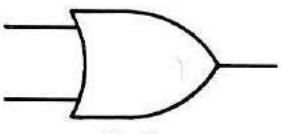


Fig. 16.

2 will give you logic 1 output. The logic symbol for an "OR" gate is shown in Fig. 16, and its operation is outlined in Fig. 17.

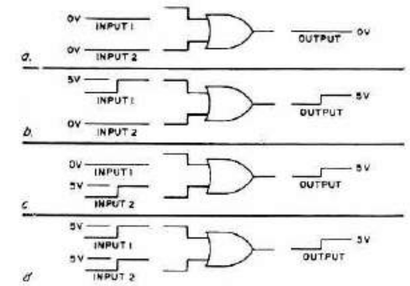


Fig. 17.

One digital IC which uses the "OR" gate is the 7432 quad 2-input OR gate. This chip has 4 "OR" gates on one IC. Each gate can be used separately; the layout of the chip is shown in Fig. 18.

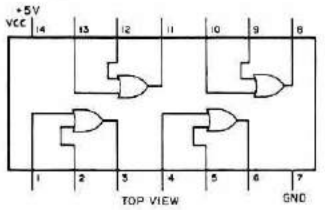


Fig. 18.

As with the other ICs that we have looked at, the +5 is to pin 14 and ground is to pin 7. Inputs 1 and 2 go to output 3, inputs 4 and 5 go to output 6, etc.

In some circuits it is desirable to have the two inputs with the OR function, plus the signal inversion of the NOT gate (inverter). It is easy to use the two gates together as shown in Fig. 19.

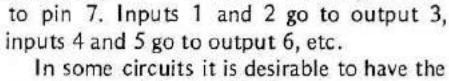


Fig. 19.

The use of the NOT and the OR gate together is also a very common combination. The combination is usually considered as one gate called a NOT-OR gate or, more commonly, a "NOR" gate. The logic symbol of a "NOR" gate is shown in Fig. 20.

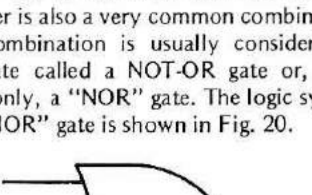


Fig. 20.

Note that the only difference in the symbol of a "NOR" gate and the symbol of an "OR" gate is that the "NOR" gate has a "o" at the output.

FIFTH: *Consider a "o" at any chip to show the signal is inverted at that point.*

One digital IC which uses the "NOR" gate is the 7402 quad 2-input NOR gate. This chip has 4 NOR gates on a single IC. Each gate can be used separately; the symbol for the chip is shown in Fig. 21. The +5 goes to pin 14, ground to pin 7, inputs 2 and 3 to output 1, inputs 5 and 6 to output 4, etc.

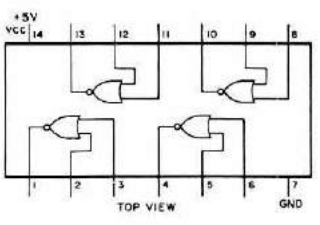


Fig. 21.

With this chip you can have the operation of a "NOR" gate, the operation of a "NOT" gate (inverter), the operation of an "OR" gate, and the operation of a "buffer". Fig. 22 shows the connections for the different gates.

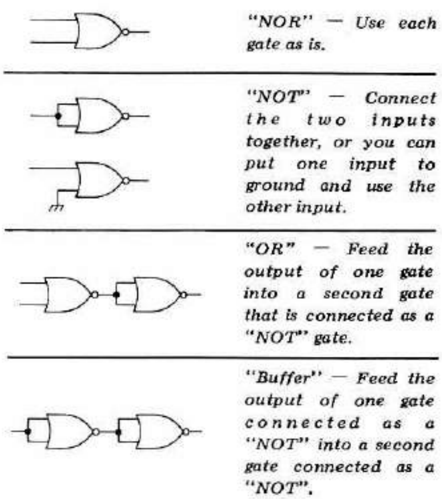
Two other IC gates that you may run across are:

1) The 4-input "NAND". There is not much difference between a 4-input gate and the 2-input gate that we looked at before. With the 4-input "NAND" gate, all 4 inputs must be at 5 volts (logic 1) to get 0 volts at the output. If any input is ground (logic 0), the output will stay at 5 volts. See Fig. 23 for the logic symbol of the 4 input "NAND".

2) The 8-input "NAND". With this gate we find that all 8 inputs must be at logic 1 for the output to be at logic 0 (ground). If any of the inputs is at ground, the output will stay at logic 1 (5 volts). See Fig. 24 for the logic symbol of the 8-input "NAND".

I have not tried to show all of the ways that TTL logic can be used or all of the different types of gates that you may see from time to time. I have not included anything on flip flops, decade counters, or any of the more complex ICs, most of which would take an article the size of this one to discuss fully.

What I have tried to do is show some of the fundamental logic gates that make up most of the TTL logic circuits. If you have an understanding of the operation of the "buffer", the "inverter" (NOT) gate, the "AND" gate, the "NOT-AND" (NAND) gate, the "OR" gate and the "NOT-OR" (NOR) gate, you are ready to start building with TTL.



"NOR" — Use each gate as is.

"NOT" — Connect the two inputs together, or you can put one input to ground and use the other input.

"OR" — Feed the output of one gate into a second gate that is connected as a "NOT" gate.

"Buffer" — Feed the output of one gate connected as a "NOT" into a second gate connected as a "NOT".

Fig. 22.

My suggestion now is to get a 5 volt power supply, and 2 or 3 7400s, and see just what they will do. Then pick a project from the pages of "73" Magazine, and try it. Don't be too surprised if it works the first time the power is turned on. If it doesn't, look back over the fundamental gate operation and I'll bet you find out why in a very few minutes.

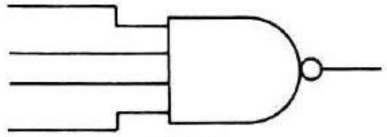


Fig. 23.

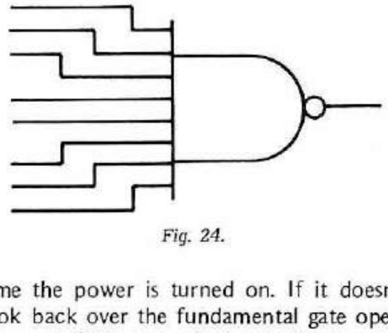


Fig. 24.

Above all, remember "Browning's Rules of Order":

1. Stay within 4.5 and 5.5 volts.

2. Do not connect two outputs together.

3. Connect a resistor in series with any input or output which goes to the 5 volt power supply.

4. Consider a "floating" input as logic 1.

5. Consider a "o" at any chip to show that the signal is inverted at that point.

. . . WB5IRY

Wayne Ledder W1EWL
8 Overlook Drive
Medway MA 02053

The Ins and Outs of TTL

The design of digital circuits using TTL integrated circuits can be much less frustrating if some simple rules are followed. I am referring to the rules that dictate how and why interconnections between the various circuits and the outside world are made. If you follow the rules, you should be able to put together a digital circuit from TTL ICs and only have to worry about wiring errors, logic goofs, and bad ICs. The information is from the manufacturers' literature and my own experiences on the bench.

Ins

The most common and most easily overlooked input of TTL circuits is the power supply. One look at any TTL circuit and you quickly know the value of the supply voltage is +5 V. For those who worry about such things, the tolerance is ±5% (military versions are ±10%). In other words, the manufacturer only guarantees proper operation of his ICs when the supply voltage is between +4.75 V and 5.25 V. This is not to say they won't work at other voltages — only that there is no guarantee.

When TTL circuits switch they generate very high frequency current spikes on the power supply lines. These current spikes traveling through the high frequency impedances of the power supply lines cause voltage spikes which can couple into other circuits and trip flip flops, clock counters, and do all sorts of nasty (and very difficult to find) things. To protect yourself from this problem these power connection rules should be followed:

1. Connect a .01 uF disc capacitor from the +5 connection to the ground connection of each IC. Locate the capacitor as close as is practical and use short leads. A miniature disc with a voltage rating of 10 volts or more is a good choice.

2. Use fairly heavy wire (I recommend #18 or larger) for +5 and ground lines and

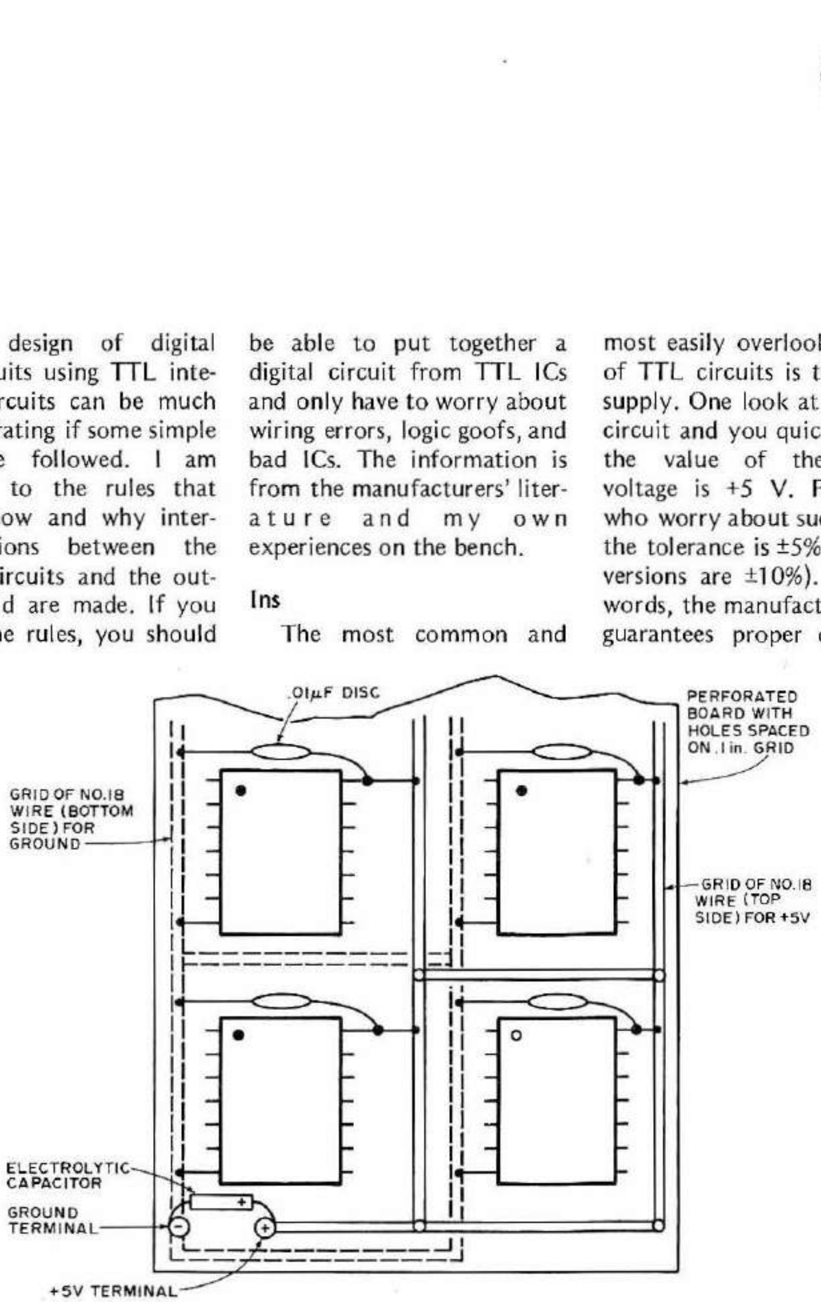

.01μF DISC
PERFORATED BOARD WITH HOLES SPACED ON .1 in. GRID
GRID OF NO.18 WIRE (BOTTOM SIDE) FOR GROUND
GRID OF NO.18 WIRE (TOP SIDE) FOR +5V
ELECTROLYTIC CAPACITOR
GROUND TERMINAL
+5V TERMINAL

Fig. 1. Illustration of rules for power supply connections.

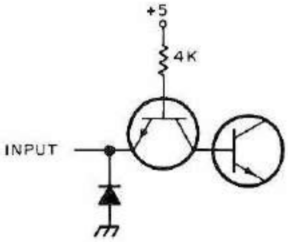


Fig. 2. Standard TTL input circuit.

arrange them so there are many connections. Try to simulate a ground plane and a +5 plane.

3. For every 20 ICs or so in your circuit put in one electrolytic capacitor from +5 to ground. Any value between approximately 4 and 25 uF and 10 volts or more rating is OK. If only one is used, try to locate it where the +5 first comes onto the board. If more are used, distribute them more or less evenly over the board.

4. Use a regulated supply for the +5 V. There are many circuits for making a regulated supply. Look at almost any article using ICs and pick one that suits your requirements.

Some of the above rules may seem obvious while others are not so well known, especially to the newcomer. Fig. 1 illustrates one method of construction employing point to point wiring following the above rules.

Next on the list of TTL "ins" let's investigate a typical input circuit as shown in Fig. 2. In order to guarantee that the transistor is turned on we must do two things. First, we must make the input voltage less than .8 V, and second we must draw out of the emitter 1.6 mA of current. In order to guarantee that the transistor is turned off we must also do two things. First, we must make the input voltage greater than 2 V, and second we may have to supply up to 40 uA of leakage current. The diode is not necessarily present in all circuits. Its purpose is to limit negative pulses on the input that may occur due to transmission line effects on long interconnections. This input characteristic is called 1 unit load (UL) and a circuit such as Fig. 2 which contains 1 UL is said to have a fan-in of 1.

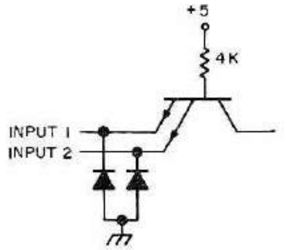


Fig. 3. Input circuit of 2-input gate.

If a second emitter is added to the circuit of Fig. 2 we have a 2-input gate configuration, as shown in Fig. 3. Each input has its own protection diode. If either input satisfies the "on" requirements the transistor will be on. Obviously, both inputs must satisfy the "off" requirements in order to turn the transistor off. By adding more emitters the manufacturers make multiple input gates. The 7430, for instance, has 8 emitters.

What about the undefined area of the input characteristic which lies between .8 V and 2 V? It is just that, undefined. This is a "grey area" where nothing is guaranteed and, except for some special circuits (discussed later), it should be passed through as quickly as possible (less than 200 ns). If the input passes through this region too slowly the output can actually break into oscillation. In fact, this is how a TTL oscillator gets started: by being biased deliberately into this "grey area" until oscillation occurs and then having the frequency of oscillation controlled by external components.

Inputs from the Outside World

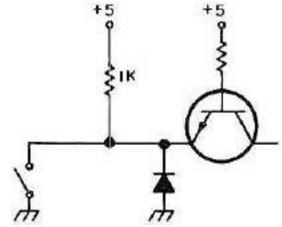


Fig. 4. Using a normal switch to drive TTL.

TTL circuits connect to themselves very nicely, but the outside world is not necessarily TTL compatible. How do you satisfy the input requirements outlined above? It is very easy to get a voltage less than .8 V and able to sink 1.6 mA: Just short the input to ground with a switch. What about the other limit? How do we get the high voltage and current source? Fig. 4 shows one way. The 1k resistor guarantees that even with 40 uA being drawn, the voltage will be above the required 2 V minimum.

When the input is relatively slow in changing you can avoid the "grey area" problem by using TTL ICs which have a special hysteresis built into them. These are called Schmitt triggers and have different switching points depending on whether the signal is positive or negative going at the time. The 7413, 7414, and 74132 are examples of this Schmitt trigger type of circuit.

When considering switches as the connection to the outside world another potential problem arises. The contacts of the switch don't close "cleanly." They actually hit and bounce apart one or more times before remaining closed. Normally this is no problem, but if you were attempting to count switch closures it would not be possible. Fig. 5 shows how to use an SPDT switch and two NAND gate elements to form a flip flop which debounces the switch. This is a very common form of circuit configuration called cross-coupled NAND gates.

Often when you are finished with a logic design you will find yourself with unused inputs to some circuits. *Never*, repeat, *never* leave any unused input to float; it will cause nothing but trouble. Even though an open input is theoretically the same as a high, in practice it is very sensitive to any kind of noise and can cause the output to change for no apparent reason.

What do you do with unused inputs? There are several choices:

1. If it will not prevent normal operation of the circuit you can ground the unused input or connect it to a high source.

1a. The high can be obtained by connecting directly to +5 V. The manufacturers don't recommend this since, if the input goes above +5.5 V, it is possible to damage the input if the current is not limited in some way. I do it all the time with no problems (yet). I recommend connecting to the +5 V connection on the chip itself.

1b. Connect the unused input to +5 V

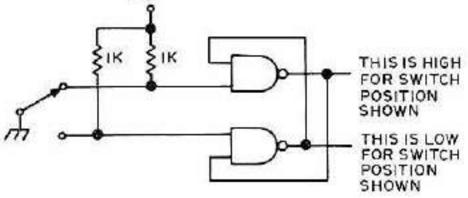

+5
1K
1K
THIS IS HIGH FOR SWITCH POSITION SHOWN
THIS IS LOW FOR SWITCH POSITION SHOWN

Fig. 5. Using cross-coupled NAND gates to debounce a switch.

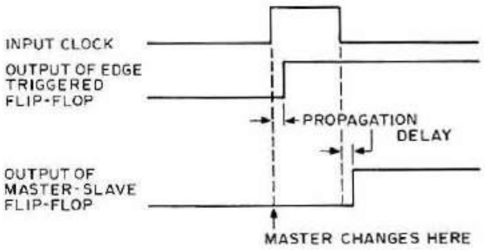

INPUT CLOCK
OUTPUT OF EDGE TRIGGERED FLIP-FLOP
PROPAGATION DELAY
OUTPUT OF MASTER-SLAVE FLIP-FLOP
MASTER CHANGES HERE

Fig. 6. Timing diagram showing difference between edge-triggered and master-slave flip flops.

through a 1k resistor. One resistor can tie as many as 50 inputs to +5 V.

1c. Use an unused inverting element and ground the input(s) to force the output high. This high output can then be used to pull up as many inputs as its fan-out is rated (see output section).

2. Unused inputs of gates can be connected in parallel with used inputs. There is no increase in load for low inputs. The total current required for the two inputs in parallel is still 1.6 mA. Actually you can tie as many in parallel as you wish and the total low fan-in will not exceed 1 UL. The high fan-in does increase, however, as each emitter may require the 40 uA leakage current. As long as the high fan-out capability of the driving circuit is not exceeded you can parallel gate inputs.

One more input characteristic of TTL circuits is worth mentioning. This is the difference between so-called edge-triggered and master-slave flip flops. The outputs of both circuits react according to the status of the control lines at the time the clock pulse occurs. An edge-triggered flip flop output reacts essentially immediately. The small delay is called propagation delay. A master-slave flip flop is more subtle. On the leading edge of the clock pulse the master reacts like an edge-triggered flip flop. The output, however, is from the slave and it does not react until the trailing edge of the clock pulse. Fig. 6 shows this difference in graphic form. The main point to remember is that a master-slave flip flop requires a complete clock cycle, not just one edge.

Outs

There are three basic forms of output circuit used in TTL. The most common form is shown in Fig. 7. This is the so-called totem pole configuration. In the low output state the bottom transistor is on and the top transistor is off. The bottom transistor sinks the current from the connected inputs. In the high state the bottom transistor is off and the top transistor is on. The top transistor now supplies the leakage currents for the connected inputs. This is the normal version. There are minor variations on this circuit but they all operate the same way.

The number of unit loads an output can drive is called its fan-out. The standard TTL IC has a low fan-out of 10 UL and a high fan-out of 20 UL. In other words, a standard TTL output can sink 16 mA (10 times 1.6 mA) and the output is guaranteed to be no higher than .4 V or it can source 800 uA (20 times 40 uA) and the output is guaranteed to be higher than 2.4 V. If you compare these output specs with the input specs you will find there is a .4 V safety margin.

There are several ICs which are specifically designed to drive larger loads. The 7437, 7438, 7439 and 7440 will all sink 30 UL. The 7437 and 7440 will also source 30 UL.

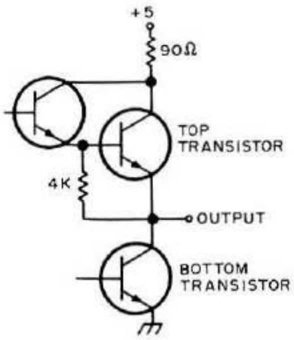


Fig. 7. Totem pole TTL output circuit.

The 7438 and 7439 belong to another class of output circuit called "open collector." In this configuration the top transistor is missing and the bare collector of the bottom transistor is brought out. With this circuit you need an external load resistor of some sort to provide the pull-up to +5 V. If the voltage rating of the IC output is high enough (the 7407 is rated for 30 V, for example), you can switch much higher voltages, and even drive relays and lamps if the current rating is not exceeded.

The open collector circuit is also used in a logic configuration known as both "wired-and" and "wired-or." Fig. 8 shows how this works. All the open collector outputs are tied to a common pull-up resistor. If any output goes low they all go low (wired-or), and the output will only be high when all the output transistors are off (wired-and, hence both terms). There is a practical limit to how many outputs you can connect together. The pull-up resistor must supply the leakage current for all inputs and all outputs when they are all off and still maintain the voltage above 2.4 V. However, it must still be large enough to limit the total current of resistor plus inputs to 16 mA (10 UL) when any output is on.

Contrary to what you may have read previously, it is OK to connect TTL outputs in parallel, provided you also connect the inputs in parallel so the outputs are doing the same thing at the same time. In fact, the manufacturers recommend this technique as one way to increase fan-out when the load is too much for one output. However, you should restrict this paralleling to elements in the same package because large transient currents are generated and can cause problems if they are not closely confined.

The third and newest type of output configuration is the so-called tri-state or 3-state output. This is a normal TTL totem pole output where both transistors can be turned off. An extra "enable" input

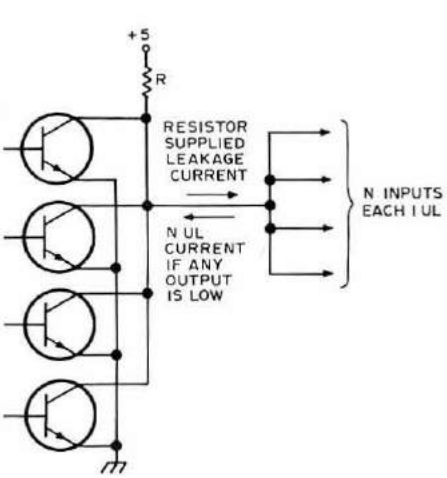

+5
R
RESISTOR SUPPLIED LEAKAGE CURRENT
N UL CURRENT IF ANY OUTPUT IS LOW
N INPUTS EACH I UL

Fig. 8. Using open collector outputs for wired-or configuration.

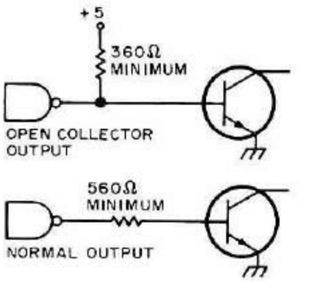


Fig. 9. TTL driving transistors.

is added to the circuit and when enabled the output functions as a normal TTL output. When disabled the output is essentially disconnected. This allows one common wire with many tri-state outputs connected to it to carry all sorts of different information (even in different directions) on a time division multiplex basis. All you have to do is enable the appropriate outputs and inputs at the proper time. This "bus structure" is very common in the world of computers and microprocessors. I have used tri-state circuits to latch 336 bits of data at one time and then output them 8 bits at a time. That is 42 different outputs on each data line.

Someplace in your design you have to connect outputs to the outside world. There are special ICs for driving displays of different kinds as this is one of the most common outputs encountered. I have also mentioned using high voltage open collector outputs. If you need more current or voltage, you can use a circuit such as shown in Fig. 9. In both versions the resistors limit the base current to a safe value. When driving an external circuit of any kind it is best to use an output dedicated to only that load. That way, if any stray signals are picked up and fed back on this line to the outside world they won't be able to couple into another input and disrupt operation. Also you should never use the output of a flip flop (this includes counters, shift registers, etc.) to connect to the outside world. It is too easy for a stray signal to sneak back, get inside the flip flop, and do weird things.

When it becomes necessary to switch large output loads, often there are transients generated that ride back in on the ground and cause stray triggers of flip flops and other nasty things. A relatively new circuit element called an opto-isolator eliminates this problem completely. Even when driving all 8 channels plus sprocket advance of a high speed paper tape punch there is no problem — and that represents 9 A at 24 V every 9 ms. The opto-isolator (see Fig. 10) consists of an infrared LED and phototransistor. The LED is driven from an open collector output and its energy is coupled optically (no physical connection) to the phototransistor, which is then used as a low level switching stage in a totally electrically isolated circuit.

The preceding rules and suggestions will not eliminate all your digital logic problems but they will greatly reduce them, especially those frustrating random ones. Remember the manufacturers only guarantee proper operation if you stay within the specs. Exceed the specs and all bets are off. ■

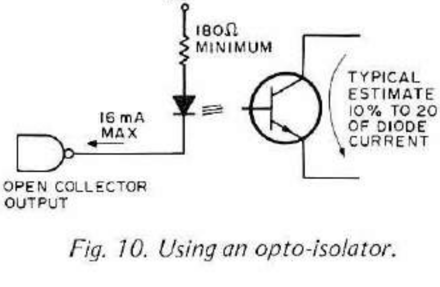


Fig. 10. Using an opto-isolator.

I/O EDITORIAL

COMPUTER PUBLICATIONS

Wayne Green W2NSD/1
Fred R. Goldstein WA1WDS

PCC — People's Computer Company, Box 310, Menlo Park CA 94025. This is a tabloid newspaper bi-monthly . . . $5 for six issues, $9 for two years . . . ran 32 pages in the latest issue. PCC is a mixture of chitchat, news of computers in the schools, game programs, news of products and services . . . emphasis is school computer use. Delightful. PCC has just announced a couple of new publications, TINY BASIC at three issues for $3, a newsletter on Tiny Basic, a new program language for tiny kids so they can write games, do math recreations and operate relays and other real time stuff. The other new publication is a series of booklets on Computers in the Classroom. Book 1, 60 pages Xeroxed, is $3.

Computer Hobbyist, Box 295, Cary NC 27511. $6 a year. This is an advanced type of newsletter and a major source for info from Hal Chamberlin, one of the leading computer hobbyist circuit designers. This is a little heavy for rank beginners . . . a recent issue had a program for generating random numbers and a cassette recorder interface circuit. Editor Stallings is doing a fine job.

Computer Notes, MITS, 6328 Linn Ave NE, Albuquerque NM 87108. While this newsletter is designed primarily for Altair owners, and much of it is far beyond the beginner to comprehend, it is a very well done and interesting publication. Costs $30 per year.

Creative Computing, Box 789, Morristown NJ 07960 . . . $8/yr. Bi-monthly. Heavy on games, complete with programs for them . . . heavy on school use of computers.

Micro-8 Newsletter, Cabrillo Computer Center, 4350 Constellation Road, Lompoc CA 93436. $6/6 issues. Aimed at the hobbyist, particularly the hardware enthusiast and the 8008 user. Tiny type and an enormous amount of info in each issue . . . sources of info, parts, equipment, names and addresses of other hobbyists. Circuits of interest. First rate source.

Microcomputer Dictionary and Guide. Matrix, Champaign IL 61820; $15.95 postpaid. Also available from Radio Bookshop, Peterborough NH 03458.

It's a whopper . . . well over 700 pages . . . and the price is right at $15.95. The term "dictionary" is not quite accurate . . . this is more like what you might expect in an encyclopedia. The definitions of computer terms are not brief; they are almost enough for you to learn about computers just from reading the book.

In a field where there is a completely new lexicon, where it is almost impossible to read articles in computer magazines or newsletters without an interpreter, where you can't even understand the computer folk when you try and talk with them, a dictionary such as this is invaluable.

It is an unfortunate fact that virtually everyone who is deeply involved with computers and has learned the new language also seems to have forgotten English. Computer folk are unable, to a man (it would seem), to write in English or even to talk it. They no longer talk with people, they interface. Without a lot of reading of *73* articles (many of which are not even written yet) you will be plain high and dry trying to fathom what these strange folk are trying to say in their weird new language.

My Computer Likes Me (when I speak in BASIC) by Bob Albrecht. Dymax, Box 310, Menlo Park CA; 64 pp., $2.00.

BASIC (A Self-Teaching Guide) by Robert L. Albrecht, LeRoy Finkel, and Jerald R. Brown. Wiley, New York; 324 pp., $3.95.

The most popular computer language for both small systems and time-sharing is BASIC. It is a powerful language, yet easy to learn, and is fast being adopted as the standard language for hobby computer program exchange. The computer novice who purchases a microcomputer will probably want to obtain a BASIC operating system for his computer, as it speeds up programming immensely.

These two textbooks are very different in scope and approach, though both are authored by Bob Albrecht of the People's Computer Company. *My Computer Likes Me* is written as a simplified approach for rank beginners, and seems aimed at the secondary school user with a classroom terminal. The approach is friendly, informal and light, with plenty of examples to show every step of the way. Only the fundamentals of BASIC are covered, with a demonstration of mathematical modeling to predict population. For a student with an innate fear of the computer and what it takes to understand one, this simple text is perfect to break down that first barrier.

Continued on page 43

Flip Flops Exposed

The purpose of this article is not only to teach the basics of flip flops, but also to help the ham builder understand the circuits which he sees every day in magazines and construction project books.

All of us try to follow the builder's explanation of operation but may fall short of full understanding due to ignorance in certain shorthand notations or unfamiliar phrases such as "the flip flop is now set," or "it operates in a master-slave configuration," or more common, "the device will be allowed to toggle."

We decide to skip the explanation and build it anyway. It works, but we miss the joy of knowing how it works. Hopefully, these intermittent knowledge gaps now can be filled once and for all.

The flip flop has an amazing talent: a memory.

This may not seem like much to you, but for a chunk of silicon it's quite an accomplishment.

This memory is the basis of all computers and counters. After all, what is counting but remembering what you had last? And the beauty of learning about flip flops is that there is no new background material with which to grapple. All that is needed is an understanding of the TTL primer in the July issue.

Take a minute and review the four basic gates and the action of the clocking input.

There are four basic types of flip flops. They are the R-S, D,T and J-K. All have two stable states, either 0 or 1. This is known as bistable. Sometimes flip flops are referred to as bistable multivibrators. Another name you might have seen is bistable latch. They are all the same thing, a form of the flip flop.

R-S Type

Fig. 1 shows the simple R-S flip flop or latch. The S means SET and the R means RESET. Other terms used are PRESET and CLEAR, respectively.

By convention, the outputs Q and $\bar{Q}$ cannot be equal. In fact, the bar over any letter means "not". $\bar{Q}$ is called "not Q".

To show the action, let's *assume* that Q = 1 and $\bar{Q}$ = 0. Because of the feedback loops the B input is 1 and the A input is 0. For the initial state in the NAND flip flop (NOR gates can also be used) R and S are always 1. So we have the initial condition, as shown in Fig. 2(a).

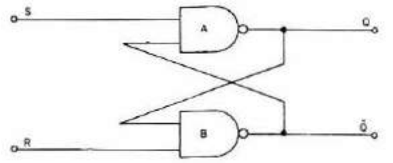


Fig. 1. R-S flip flop.

If you recall the NAND truth table, only inputs of 1,1 yield 0; all other combinations give 1.

If we drive R to 0 the following chain takes place. The R input of 0 combines with the 1 input at B to make the output at $\bar{Q}$ = 1. This 1 feeds back to the A input and combines with the S = 1 input to make a 0 output at Q. This 0 feeds back to the input at B, as seen in Fig. 2(b).

Whew! But look what happened! Our outputs are reversed! Even if we return R to its initial state of 1, the output will still remain, as we see in Fig. 2(c).

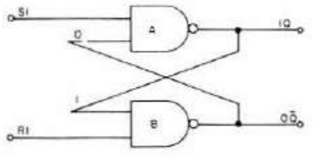


Fig. 2(a). Initial condition.

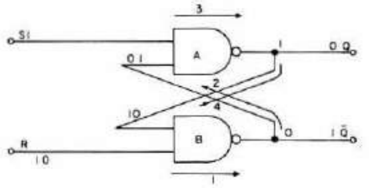


Fig. 2(b). Follow the action!

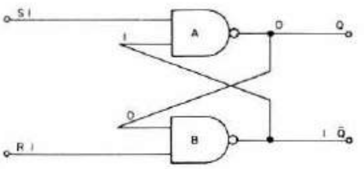


Fig. 2(c). Outputs reverse and stay, even though R is returned to 1.

Larry Kahaner WB2NEL
4259 Bedford Avenue
Brooklyn NY 11229

That momentary driving of R to 0 could be a push-button or a clock pulse.

Now if we take our final condition of 2(c) and impress S to 0, the outputs will flip around again. And they will stay, even if S is returned to 1. See Figs. 2(d) and 2(e).

The name flip flop is thus shown to be quite appropriate.

A flip flop is said to be in the SET condition if the Q output is 1. The RESET condition exists if the $\bar{Q}$ is 1.

Fig. 2(a) is a SET state and 2(c) is a RESET condition. The rule: If a flip flop is SET, driving RESET to 0 will change the output. If it is RESET, driving SET to 0 will change the output.

The symmetry is beautiful. If it's SET, reset it. If it's RESET, set it.

But what happens if both R and S are impressed to 0? Well, that is the problem with the R-S latch. The answer is ambiguous. Since you never know which 0 came first, the output is unpredictable. Try it, and you'll see the fun.

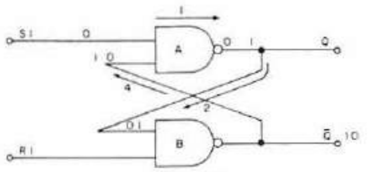


Fig. 2(d). S goes to 0 and outputs reverse.

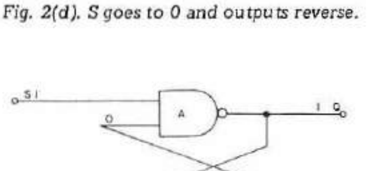


Fig. 2(e). Outputs stay reversed, even if S is returned to 1.

A schematic diagram of a flip flop may show the actual gates or it may show a configuration such as Fig. 3(a). A personal favorite is shown in Fig. 3(b).

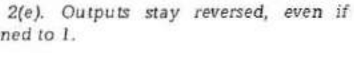


Fig. 3(a). R-S flip flop.

Either way, it indicates that something like what is shown in Fig. 3(c) may have been done with (for instance) the 7400 chip.

Not all flip flops will be made from chips containing *only* simple gates. Some ICs have flip flops and *other* elaborate circuits within one package.

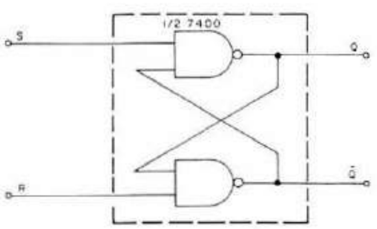


Fig. 3(b). Flip flop showing chip it came from.

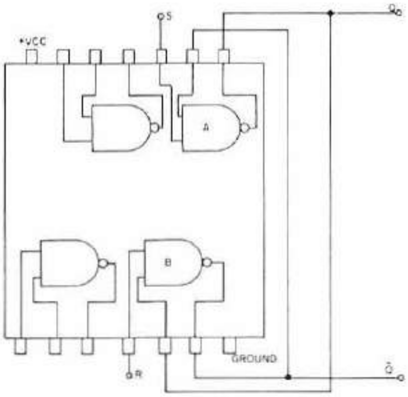


Fig. 3(c). 7400 wired in R-S latch configuration.

The R-S latch can also be made using NOR gates (Fig. 4). In this case the inputs are held low (0) and driven high (1) to start the flip flop function. It's just the opposite of the NAND gate flip flop, where inputs are held high (1) and driven low (0) to activate.

The NOR flip flop is not used much. It is easier to use NAND gates. They are also cheaper and more available.

In the NOR flip flop inputs of 1,1 are not allowed, just as 0,0 inputs are not allowed in the NAND configuration.

Clocking

Two types of logic circuits are used. One is synchronous and the other is non-synchronous or asynchronous.

Asynchronous operation exists when circuits are operating independently of each other. Each individual circuit has its own input and it responds to these inputs.

Synchronous operation relies on a common input such as a clock to feed all the circuits in the system. All functions rely on the clock.

If we add a clocking input to our R-S flip flop, two additional NAND gates are used. These gates insure that the latch works only when the clock pulse (1) is present. In Fig. 5 the input of C will be 0 only if the A input is 1 and the clock pulse is 1. Remember that the R-S configuration latches only on the 0 drive – and we have supplied it.

D Type

The second type of flip flop is the D or delay type. Since we still have the problem of the SET and RESET inputs being the same (causing an unpredictable output), the best we can do is to insure that we don't have the same inputs at the same time.

In Fig. 6 the inverter (E) negates all inputs; 0 becomes 1 and 1 becomes 0. We employ it to feed the B input. By placing it there while feeding it *and* A from the same input, we can be certain that the signals reaching the gates will be different.

Any signal going to A will remain the same and those passing through the inverter to B will change.

The D type can also have PRESET and CLEAR controls. They are superior or overriding functions. No matter what is going on, commands on these controls have priority. When either of these inputs is present (0), the output will go to 0 or 1 depending upon which function is employed.

In schematic diagrams, the letter "D" is the only indication of the flip flop type.

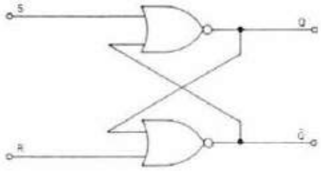


Fig. 4. NOR gated flip flop.

IC types which are D flip flops arê 7474 (contains 2 flip flops with PRESET and CLEAR) and 7475 (contains 4 flip flops without PRESET and CLEAR).

T Type

The toggle or T type does what its picturesque name implies – it toggles.

The logic is such that the output will change regardless of what it was prior to clocking. But this only happens if the clock is fast enough. If it is not rapid, the output will change state and then return. It will forever change and change back again.

Toggle flip flops have been assigned mixed jobs. When a clock pulse is applied, the output will change once every input cycle. Therefore, it completes one output *cycle* (not just change) for every two input cycles. This gives a divide-by-two property which is used in counters and calculators.

Another use is in random output devices like "electronic dice" and "heads-tails" circuits, which electronics magazines are so fond of printing once a year. Since it is unknown where the circuit is toggling at the moment, a stop-toggling command will produce a random output.

J-K Type

The J-K flip flop is very widely used. The inputs J and K correspond to S and R. The gates 1-4 are the master section and 5-8 the slave section. Gate 9 is used as an inverter. (Although not shown, a constant input of 1 is kept on the other input. It will combine with the other input to invert it. 1 and 1 yield 0; 1 and 0 yield 1.)

Both sections are synchronous (use the same clock pulse) and are activated by a pulse of 1.

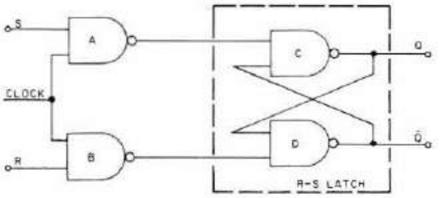


Fig. 5. Synchronous clocking of R-S latch.

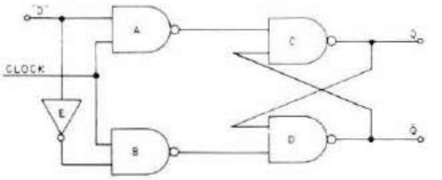


Fig. 6(a). D type flip flop.

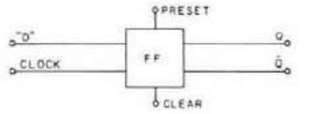


Fig. 6(b). D type with PRESET and CLEAR.

If the clock is 1, the master will see this and activate. However, the slave will see the inversion of this (0) because of gate 9, and not activate.

The feedback via lines a and b from slave to master will be valid until the clock pulse is 0. Then the slave will energize due to the inversion of this pulse from 0 to 1. But now the master is disabled because it sees the 0 straight from the clock.

The information is passed from master to slave. The next pulse will feed the output of the slave back to the master and so on.

A sequence of four steps takes place.

1. Isolate slave from master.
2. Enter information to master.
3. Disable master.
4. Transfer information from master to slave.

All this takes time. That is a useful property. A J-K can be used for storage of information while another circuit is doing something else. The two informations can then rendezvous at the proper time. The J-K also solves the same ambiguous input problem of the simple R-S. It's pretty good as a toggle, too.

Some examples of J-K flip flops are the 7470, 7473 and 7476. Each has a unique property such as number of flip flops, different voltage and frequency ratings, and different controls.

How the Flip Flop Counts (or, Here's the Part We've All Been Waiting For)

We have seen that it takes two pulses to return a flip flop to its initial state. If we start with Q as 1, it will take two pulses to make it 1 again. The memory remembers the first pulse and waits for the next.

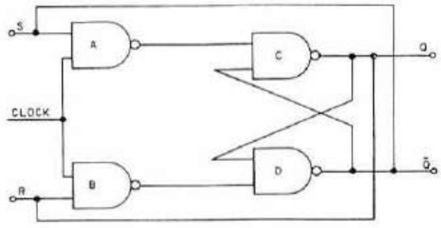


Fig. 7(a). T type flip flop.

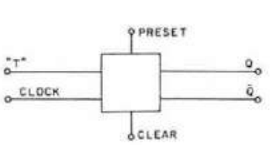


Fig. 7(b). T type with PRESET and CLEAR.

Therefore, if we monitor the output of Q we can tell when the input has completed two pulses. Since each flip flop counts by "twos", placing them together will allow the first to count by two, the second by four, the third by eight, and so on (Fig. 9).

Think of each flip flop as having two stages. Initially, all inputs and outputs are 0. Pulse 1 will energize a_1. Pulse 2 will return the first flip flop to its initial condition and store a pulse at b_1. Pulse 3 activates a_1 again. Pulse 4 flips section A and passes a pulse to b, where it combines with the pulse already there to flip section B. Our monitor at the output of B rings, buzzes or lights, and we know we have counted to 4.

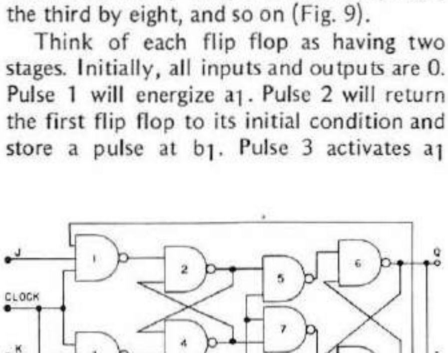


Fig. 8(a). J-K flip flop made from R-S latches.

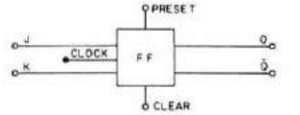


Fig. 8(b). J-K type with PRESET and CLEAR.

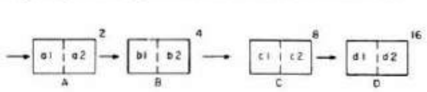


Fig. 9. Flip flop counter.

After 16 pulses the entire counter is returned to 0. We have counted to 16!

Experimentation

Many readers have asked how to convert this type of "pure" knowledge into practical application. The IC mystique still lives! Just experiment. Buy some ICs and hook them up. See what happens when you do "this" or "that". There is no great mystery. They will work – really!

Here are some suggestions to help you play around.

1) Get a 5 volt power supply. It's cheap if you build one. The October, 1974 issue of *73* has a fine one. If you like printed circuits, Radio Shack has a board which is almost the same (except for part values). Parts are very cheap from the advertisers in *73*.

2) Monitor the output with some light emitting diodes (LEDs). They tell you if the output is high (1) or low (0).

3) Get data sheets on ICs that you wish to work with. They have the circuit diagrams of the internal gates so you know how to hook them up. They also give 0 and 1 values.

4) Never be embarrassed to return non-working chips. The same mass production techniques which brought down the price also brought down the quality. Don't get worried, though: *Most* ICs *will* function properly, but if they don't, return them. I have *never* had a store *not* replace a non-working chip. Even the mail order houses exchange them without a hassle.

5) New breadboard kits have been arriving on the markets. I haven't tried them, but they look good.

6) ICs are rugged. Don't be afraid of them.

. . . **WB2NEL**

Those Exciting

Joseph C. Fowler
GTE Lenkurt Demodulator
Assistant Editor

Data processing systems have revolutionized our world, allowing vast amounts of information to be stored, exchanged, updated, and utilized in ways undreamed of a few years ago. A chain of department stores can be tied together, for example, by a data network making current inventory, personnel and credit information instantly available at widely separated locations; a railroad can control activity at its switchyard from a control center hundreds of miles distant; switching functions within a telephone company office can be accomplished rapidly and reliably in accordance with stored program instructions. The information involved in each of these examples is different, but the underlying processing principles are the same.

A digital computer or data processor of any type is basically a stored-program machine, in which a memory facility holds a set of operating instructions — the system program. Information is put into a digital format and fed into the machine, which retains it in a data memory. The instructions in the program memory, which are also in digital form, tell the processor what problem is to be solved, or function performed, related to the input data. When the operations demanded by the program memory have been completed, the data is fed out to be utilized in some manner. In practice, program instructions are often stored within the same memory facility as the data; in this way, the program can also be changed if desired.

Despite surface differences, the ways in which memory facilities receive, hold, and feed out information are all based on either sequential or random access principles.

Sequential Memories

The sequential, or serial, memory method requires that the data bits comprising the information be arranged in a particular order. Data stored in this manner — including both programmed instructions and input information — is retrieved strictly in accordance with its position in a time sequence.

A simple example of sequential memory storage is paper tape, which uses the presence or absence of holes to indicate the condition of a data bit; the combined states of several bits identify a particular digit (see Fig. 1). The tape is moved through a sensing device to convert the hole patterns into a series of pulses for electronic processing. Each piece of information is retrieved as it passes the sensor. Another form of sequential memory storage is magnetic tape, which stores information as magnetic flux variations corresponding to the 1s and 0s of digital data.

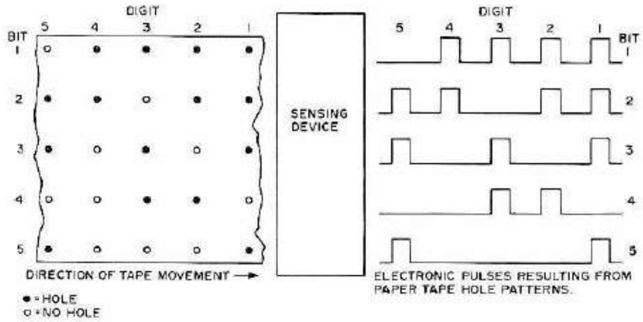

DIGIT
5
4
3
2
1
BIT
1
2
3
4
5
SENSING DEVICE
DIRECTION OF TAPE MOVEMENT →
● = HOLE
○ = NO HOLE
ELECTRONIC PULSES RESULTING FROM PAPER TAPE HOLE PATTERNS.

Fig. 1. Paper tape is a form of sequential memory storage, in that the bits identifying a given digit can only be retrieved during their allotted sensing time, which may cause delay in finding desired data.

Sequential access memory systems, including punched cards and magnetic disks as well as tapes, have been widely used in the area of mass computer memories. They typically contain program instructions and must be physically introduced into the processor system — threaded through a sensing device, for example — so that input data can be operated upon. They may also be used to retain the data for future processing. These devices provide a permanent storage capability since the cards, disks, and tapes can be removed and filed for repeated use, and they have non-destructive readouts (i.e., data does not have to be re-entered every time it is used); however, they have some shortcomings which limit their usefulness in high speed memory applications.

For example, sequential access can be a relatively slow process because a large amount of irrelevant data may have to be scanned before the desired bits are found. This delay may be from milliseconds to minutes, which is much too great for many of the uses to which modern data processing equipment is put. Additionally, these sequential

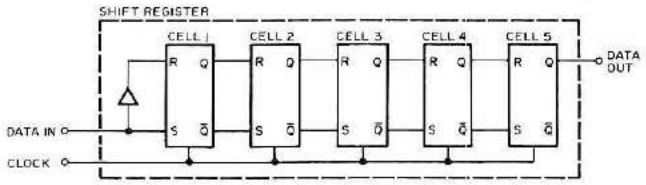


Fig. 2. The shift register, which stores data for as long as it takes for the clock to move it through each cell, is typical of semiconductor sequential memory systems.

Reprinted from the ***GTE Lenkurt Demodulator***, September, October, 1975.

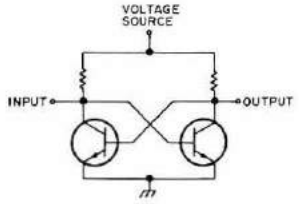


Fig. 3. The bistable multivibrator, or flip flop, has two stable states, making it an ideal digital data memory storage cell.

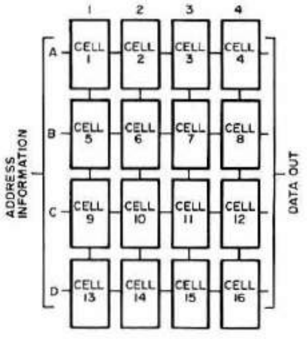


Fig. 4. A random access memory (RAM) is a matrix of memory cells, any of which can be accessed without regard for any other cell.

mass memory systems require a mechanism to move the data-carrying medium past the sensor; this device is of necessity completely mechanical, and is subject to the adjustment and maintenance considerations which apply to all such devices.

Memory Chips

-- RAMs, ROMs, PROMs, etc.

Temporary Memories

Cards, tapes and disks continue to play an important role as high density, long-term mass memory storage elements in such applications as personnel record maintenance, inventory control, and retention of performance data for comparison with future achievements.

For low capacity, temporary memory storage, however, structures composed of semiconductor devices have become dominant in recent years. Temporary data storage facilities are used in such areas as office equipment (calculators, etc.) and immediate-use or "working" memories wherein a data processing device can hold the data with which it is dealing at any given time. It is not uncommon for a data processing system to use tapes and similar elements as permanent, high volume program storage facilities, and semiconductor structures for the "working" memories in which the stored data is operated upon.

Temporary memories are also widely used in data processing terminals, which serve as remote input/output units for a large central computer and may be in any number of forms, from a simple typewriter keyboard to a small computer. Terminals provide a means of encoding data for manipulation by a central computer, and decoding it for use by a human operator.

Interconnection of central computer and remote terminal is commonly made over telephone lines through an interface unit called a modem, or data set. The data set may also contain a temporary memory facility which allows it to hold data and either condition it for transmission over the lines or prepare received data for application to the processor.

The semiconductor devices used in temporary memory structures are typically arranged as groups of individual units called memory cells, each of which stores one bit of information as either a logic 1 or logic 0. A cell may consist of as little as one transistor-capacitor combination, or it may be a complex arrangement of several components. But, whatever its composition, it has at least two states that can represent digital data bits.

Shift Registers

A shift register is a device for the temporary storage of digital information; when a shift, or clock, pulse is applied, the register accepts new data and moves every stored bit one step toward the output.

Fig. 2 shows an example of a semiconductor shift register containing five memory cells, each of which is a solid state reset-set (R-S) flip flop circuit. Data applied to the register input in a digital form is stored and shifted from cell to cell in accordance with the clock cycle rate. A logic 1 on the S input of cell 1, for example, will set the flip flop to the 1, or "on," state if a clock pulse is present at input C, thus storing the digit. On the next clock pulse, the stored bit is shifted out of cell 1 to set the second cell to the 1 state, and a new digit is fixed in the first cell. This procedure continues through the register, with the output of the final cell being shifted out for data processing. The clock frequency controls the rate at which the shift register stores and feeds out data bits, with each bit being delayed between input and output by as many clock cycles as there are cells in the register. Since the storage and retrieval are done on a first-in, first-out basis, the shift register is a sequential memory storage device.

Digital data can also be handled by a random access process. This does not mean, of course, that no orderly procedure is involved; it means, rather, that information can be stored in a particular memory cell, or location, and retrieved without regard for any other location.

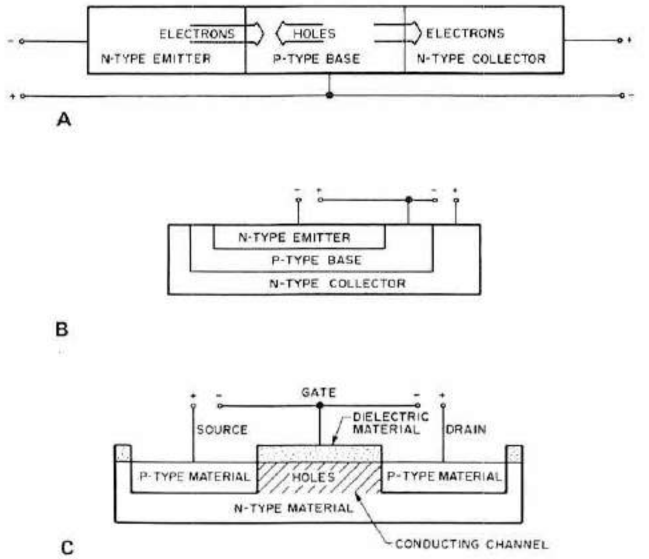


Fig. 5. Bipolar and unipolar transistor structures are the most widely used semiconductor memory cell components.

Random Access Memory Systems

A random access memory (RAM) can be defined as a structure in which any data bit can be stored (written) or retrieved (read out) in any order.

One of the most basic ways to create a memory cell is through the use of the bistable multivibrator, or flip flop. As shown in Fig. 3, such a memory cell may consist of only two transistors, two resistors, and a power source. In this cell, one or the other of the transistors is always conducting, holding the other one off. When an external signal forces the off transistor into conduction, the initially on transistor turns off and remains in this condition until another external signal resets it. The flip flop, therefore, has two stable states which can be used to store information in the form of logic 1s and 0s.

A RAM is essentially a matrix of such memory cells, with each cell identified by a unique code, or address. The data processing equipment can retrieve a bit of information by addressing the proper location. Because of the matrix structure, the time required to locate any given bit is approximately the same as that required to locate any other bit. For example, in Fig. 4 the digit stored in cell 1, at location A1, could be available at the data output in almost exactly the same time as the bit in cell 16, location D4. This rapid access to information makes the RAM ideal for application as a temporary storage facility.

Random access memory structures are of two basic types: read/write and read only. A read/write RAM is programmable; that is, data can be entered into, changed, and removed from the memory at any time. A read only memory (ROM), however, has certain data patterns fixed into it, usually during the manufacturing process. In such a structure, information can be read out – it will always perform the same function – but the stored program does not change. Because the data pattern is fixed, a ROM retains its program regardless of circuit power considerations; that is, it is a "non-volatile" memory device. A read/write memory, however, needs a constant source of power to remain in operation; if power is removed, the semiconductors stop conducting and the stored information is lost, so the read/write RAM is considered to be a "volatile" device.

Although it is not technically accurate to do so, common usage has led read/write memories to be referred to simply as RAMs, while read only structures – which are, in reality, a type of RAM – are designated ROMs, and this discussion will follow the same nomenclature.

Bipolar and Unipolar Transistors

The two most widely used devices in memory matrix design today are the bipolar and unipolar, or field effect, transistor. Each can be easily realized as an integrated circuit (IC) component, and they are readily adaptable to virtually any circuit configuration.

Essentially, a bipolar transistor is a semiconductor device whose conductive properties depend upon both majority and minority carriers; that is, current flows in a bipolar transistor because of the simultaneous movement of both positive and negative charges. Negative charges predominate in n-type semiconductor material because there is a surplus of free electrons within the material's atomic structure; p-type material, however, has a shortage of free electrons. The regions in which the electrons would normally exist act as positive, mass-bearing charges called "holes"; p-type material thus maintains an excess of positive charge carriers. The common transistor is a general type of bipolar device, since its current flows due to hole and electron movement. Fig. 5A shows the normal flow pattern within an NPN structure (Fig. 5B shows a bipolar NPN structure as an integrated circuit). The forward-biased emitter-base junction allows electrons to be injected by the emitter into the base region. Within the base, the greater part of the current flow is caused by holes combining with the excess electrons. The reverse bias of the collector-base junction allows electrons to pass into the collector region; because there are two n-type regions, electrons are the majority carriers, although the simultaneous action of the minority carrier holes is indispensable.

The field effect transistor (FET) is a unipolar device, in that its current flow is the result of the movement of only one type of carrier. In what is called the p-channel FET, holes are the majority carriers, while the carriers in an n-channel FET are electrons.

Fig. 5C shows a p-channel FET structure operating in the enhancement mode, which is the most common operating mode for FETs. In this mode, there is no conduction within the device when the gate voltage is zero; the other mode of operation is called depletion, wherein the semiconductor device is always conducting and requires a proper gate-to-source voltage to turn off.

When the gate in Fig. 5C is made negative with respect to the source, it creates an electrostatic field which attracts holes from the n-type semiconductor material toward the area directly below the gate dielectric material. Initially, this n-type area has a surplus of electrons, but as the holes are drawn into it, the electrons are neutralized. At some gate voltage, the holes become dominant and a current-carrying channel is produced between source and drain in which holes are the majority carriers. Because the gate is electrically isolated from the rest of the structure by the dielectric material, there is no current flow into it; the channel, which allows current to flow between

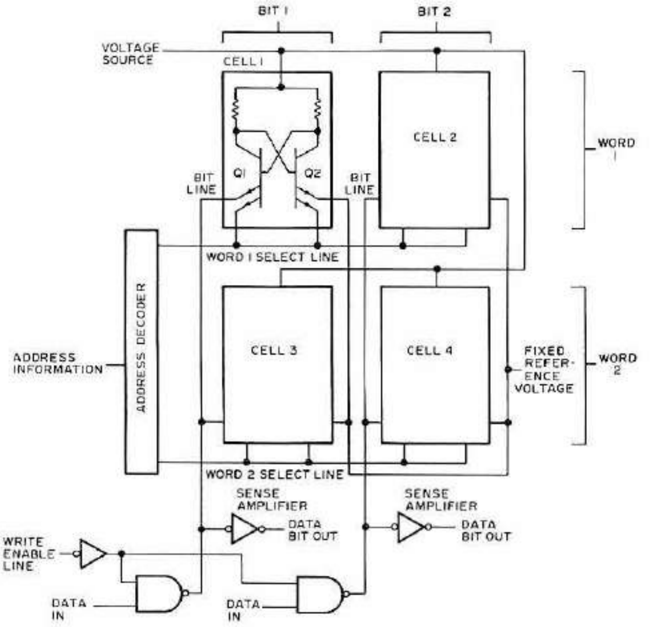


Fig. 6. A 2-word, 2-bits-per-word memory array utilizing multi-emitter bipolar transistor structures.

source and drain, is created and maintained by the electrostatic field.

The need to provide more electronic function in increasingly small areas has resulted in a great variety of miniaturized circuits. Bipolar transistors, for example, can be realized as discrete items, such as those seen in various entertainment products. In data processing applications, however, the large number of components required to produce a memory matrix makes the use of such bulky devices impractical; a circuit board with several hundred discrete transistors mounted on it — which is what a memory matrix would require — would be too unwieldy to be of any real use.

Bipolar Transistor RAMs

Integrated circuit techniques allow quantities of circuit elements to be realized in a small space; these techniques have been used to produce bipolar transistor memories of various microminiaturized sizes and densities.

The basic storage element in these matrices is the bistable flip flop, which appears in many circuit variations to meet different application requirements.

A bipolar RAM cell which has been widely used is the transistor-transistor-logic (TTL) type, exemplified by the multiple-emitter circuit (see Fig. 6). In this case, the data processor applies an address code to a decoding circuit; the decoded address raises the voltage on the correct word select line (places it at a logic 1 level), preparing the cell for the read or write function. A logic 1 bit can be written into the cell, for example, by placing the write enable line at a low voltage (logic 0) level while the data bit input is logic 1. This causes the bit line to be low, turning Q1 on and Q2 off, a state which represents a logic 1 within the cell. Once the write function is complete, the address changes, the word select line returns to a logic 0 state, and Q1 remains on. To read out the stored digit, the address raises the word select line level and the write enable line is held high (logic 1), allowing read current representing the value of the cell's contents to flow into the appropriate sense amplifier for output to the data processor. Because the read process does not change the state of the flip flop, the stored information is not lost and the cell is considered to have a "nondestructive" readout capability.

The 2-word, 2-bits-per-word memory shown in Fig. 6 is, of course, limited in its application. The same addressing, reading, and writing functions, however, are performed in bipolar RAMs containing several times the number of cells. A typical example of expanded capacity is a single integrated circuit capable of storing 16 words of 4 bits each, for a total of 64 bits on one tiny silicon chip. GTE Lenkurt uses nine such chips in both its 262A and 262B data sets. The bipolar RAMs comprise a data memory, in which input data is held to be operated upon. Because of the read/write capabilities of the RAMs, the data being processed can constantly be updated and changed.

Major considerations in memory design include the speed with which a cell can be made to change state (access time) and the amount of power dissipated by the cell's components.

Operating in a saturation mode — in which the "on" transistor constantly conducts the maximum possible current — the TTL-type memory cell requires some amount of time to drive the transistor out of saturation before a change of state can occur. This delay is only on the order of nanoseconds, but is enough to concern circuit designers. In addition, the saturation mode consumes relatively large amounts of power. Two of the more successful configurations developed to overcome these disadvantages are the diode coupled and emitter-coupled logic (ECL) cells.

Diode Coupled RAMs

Two gating diodes are used to control conduction in a diode coupled cell (Fig. 7). In integrated circuits, these diodes are frequently "hot-electron," or "Schottky barrier," devices, which become forward-biased at lower voltages than conventional diodes.

If the state of a cell must be changed to store a bit, the address decoder causes the voltage on the word select line to be forced low, while the voltage is raised on the bit line associated with the transistor to be turned off. Referring to Fig. 7, in which Q2 is hypothetically to be turned off, raising the bit line B voltage and dropping the word select line (effectively making it more negative) draws additional current through R4, increasing the base voltage on Q1 to the point at which it begins conducting. The cross-coupling of the transistors then causes Q2 to turn off, thus effecting the cell's change of state.

Reading the stored digit out of a diode coupled cell also requires that the word select line be forced low, but in this case there is no voltage increase on either of the bit lines. The combined effects of the lowered word select line and a bias network cause the diode associated with the "on" transistor to be forward-biased, causing the diode to conduct. Since the diode associated with the "off" transistor is reverse-biased, a differential voltage develops between the bit lines. A sensing circuit determines the cell's logic state from this voltage.

Read current in a diode coupled cell is greater than standby current, which flows when the cell is storing a bit without being addressed, but is substantially lower than write current. Because of this, the voltage developed across the load resistors during the read operation is not great enough to change the cell's state and the readout is a nondestructive process.

Since standby current is lower than read current and is present a greater percent of the time, overall power consumption in a diode coupled cell is lower than that of a TTL device.

ECL RAMs

The structure of emitter-coupled logic (ECL) memory cells closely resembles that of TTL cells, but biasing techniques are used to keep the transistors out of saturation. This allows the ECL storage element to change state very rapidly; the greatest advantage of ECL over other semiconductor memory configurations is that it has the shortest access time of all. Reading and writing processes are accomplished in essentially the same manner as for TTL, but at a greater speed. Because it constantly draws high current, however, an ECL memory cell has even higher power consumption than TTL, a fact which does impair its usefulness in certain applications.

MOS Technology

One of the major objectives in semiconductor memory design has been to incorporate as much capacity as possible in the smallest area. The greatest size reductions have been achieved with metal oxide-silicon (MOS) techniques, which produce field effect transistor (FET) structures that are considerably more compact than the bipolar integrated circuits (ICs) previously discussed.

An MOS FET is formed by depositing an insulating metal oxide – most often silicon dioxide – on a chip of silicon. Etching processes then remove the oxide from selected areas of the chip, exposing the substrate at source and drain locations while leaving the gate region insulated. Further processing establishes n- and p-type areas within the substrate.

The size reduction possible with MOS techniques allows a much denser memory array to be produced within a given space than is possible with bipolar devices; there are also substantially lower power requirements and reduced packaging costs.

Storage cells composed of MOS FETs may be of either a static or dynamic nature. A static cell retains its stored data as long as power is supplied to the circuit; a dynamic cell depends upon capacitive charge storage to hold its data, and must receive a "refresh" input to counteract the effects of leakage.

Static MOS RAMs

The basic static MOS RAM cell is a bistable multivibrator (see Fig. 8) closely resembling the bipolar flip flop used in TTL memories. In an MOS flip flop, however, transistors serve not only as cross-coupled inverters (Q3 and Q4), but also as load resistances (Q1 and Q2). Electrical isolation of the FET gate results in a very high input resistance which can be controlled by the gate voltage. A large-value resistor can thus be produced by an FET in a relatively small space compared to a conventional resistor.

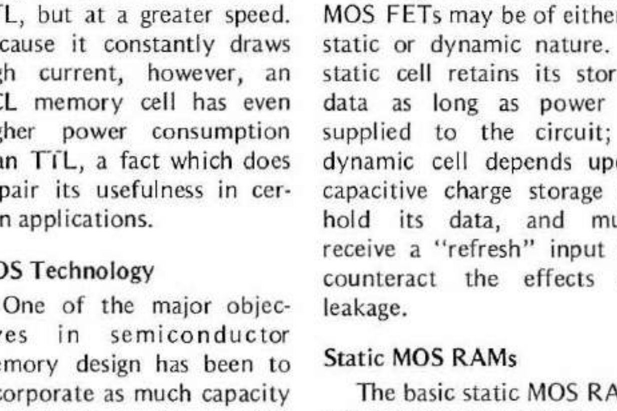


Fig. 7. A diode coupled memory cell uses gating diodes to control conduction and reduce power consumption.

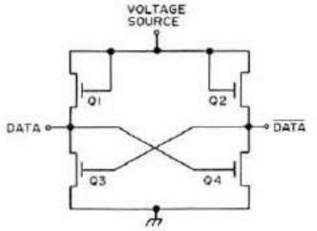


Fig. 8. The heart of the static MOS RAM storage cell is the bistable flip flop composed entirely of field effect transistors (FETs). The logic level at one terminal is always the complement of that at the other.

Since only one of the cross-coupled inverters conducts at any given time, the cell has two stable states which can be used to store information in the form of logic 1s and 0s. The state of the cell is determined by external address and data signals. The cell's state remains constant unless changed by an external signal, so no refresh action is required and the circuitry needed to support the operation of the cell is simplified.

A static MOS RAM storage unit, however, contains a minimum of four transistors, so it occupies a considerable amount of space on a silicon chip and consumes a relatively large amount of power. Because of these disadvantages, static devices have been largely replaced by dynamic MOS RAMs.

Dynamic MOS RAMs

The basic storage element in a dynamic MOS RAM cell is a capacitor, which holds and releases a stored charge in response to read and write commands. While the capacitor could be an external device, it is much more common for dynamic RAMs to utilize the capacitance existing between gate and source of the MOS FET itself. This capacitance is due to the isolation of the gate from the rest of the structure by a dielectric material. Charging the gate-source capacitance sufficiently to turn the transistor on represents a logic 1 state in most applications, while a lower charge or no charge at all serves as a logic 0.

Inevitably, as with all capacitive devices, the charge stored in the gate-source region drains off due to leakage current. If the charge is allowed to deteriorate too much, the data bit is lost, so some means must be provided to periodically restore, or refresh, the charge; a common requirement is that every cell in a memory matrix be refreshed every 2 milliseconds. Circuits to accomplish this are included in dynamic RAM designs, as are address decoding circuits.

The operation of a typical dynamic MOS RAM cell can be illustrated with the 3 transistor cell shown in Fig. 9. In this case, information is stored as a charge in the gate-source capacitance (C_G) of transistor Q2. To write a data bit into this cell, an address decoder produces a write select signal, activating transistor Q1 and allowing data on the write data line to be transferred to the storage element. Depending upon the state of the data input, C_G either charges or is discharged. When the write select signal is removed at the end of the write cycle, the bit is held in the cell.

At the beginning of a read cycle, both the read and write data lines are preset to some voltage. When the address decoder produces a read select signal, Q3 is ready to begin conducting. If the charge on C_G is sufficient (logic 1), Q2 turns on and current flows through Q2 and Q3, reducing the voltage on the read data line. With no charge on the capacitance, Q2 and Q3 remain off and the read data line stays at its preset level. Because of the gate isolation, C_G is in the same condition (charged or discharged) at the end of the read cycle as at the beginning, making the read process "nondestructive." An output amplifier senses the state of the read data line and determines what cell condition would produce it (a low-level line generally indicates a stored logic 1) for application to the data processor.

Refresh

Refresh of the stored digit in Fig. 9 is accomplished through a clocked amplifier connected between the read and write data lines. Control circuitry provides the timing necessary to keep the refresh cycle separate from the read and write operations.

The refresh process involves reading out the stored digit and writing it back into the cell. To do this, both data lines are preset at the beginning of the refresh cycle. A read select signal is then produced, transferring the bit to the read data line in the same manner as the normal read operation. The refresh amplifier inverts the condition of the read data line and applies it to the write data line. A write select signal then replaces the read signal and the data present on the write data line is entered into the memory. If, for example, a logic 1 (maximum charge) is stored on C_G, the read data line is forced low (logic 0) when the read select signal forces Q2 and Q3 into conduction. The refresh amplifier inverts this and applies logic 1 to the write data line; the presence of the write select signal causes this data to be written into the cell as a refreshed bit. With no charge (logic 0) on C_G, this sequence is repeated, with a logic 0 appearing on the write data line to ensure that the capacitance is not charged by stray circuit currents.

In Fig. 9, timing from the control circuitry allows a single amplifier to serve an entire column of cells. One alternative configuration uses a common read/write data line. This lets the cell form a loop within itself and thus eliminates refresh amplifiers.

ROMs

A read only memory (ROM) is a data storage facility into which information is normally written only once. After this entry, a ROM always produces the same output when addressed.

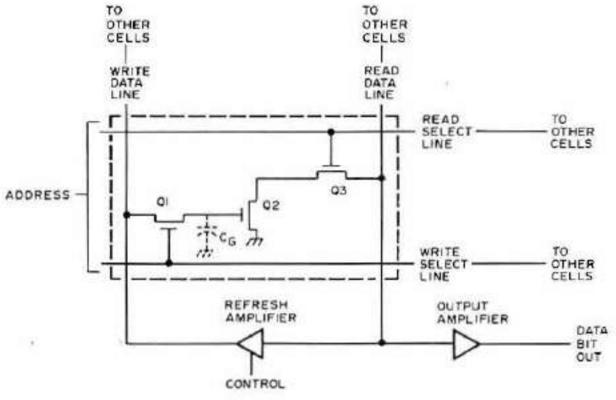


Fig. 9. A dynamic MOS RAM cell stores data in the gate-source capacitance of one of its transistors.

The difference between the read/write RAM and the ROM can perhaps be best illustrated with the example of the pocket calculator. In almost all calculator designs, a RAM matrix serves as a "working," or data, memory and ROMs are used for input/output interface, timing control and program storage (see Fig. 10).

Each key on the calculator keyboard is identified by a unique binary number; all of these numbers are permanently fixed in the ROM encoder so that, when a key is pressed, the corresponding binary number appears as the encoder output. If a digit key is pressed, the bits comprising the number are written into the RAM data store. Function key (addition, subtraction, etc.) numbers are applied to the ROM program store as addresses. In the program store are contained instructions for each function; when an address is presented, the proper instructions are read out of the ROM, leading to performance of the desired operation upon the data held in the RAM. When the function has been completed, the result is read out and applied to the decoder, which puts it into a form suitable for display.

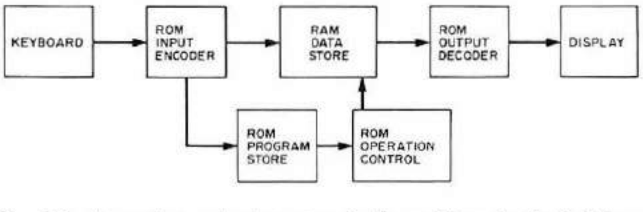


Fig. 10. A pocket calculator typically utilizes both RAM and ROM facilities to process input data.

The basic ROM structure is a matrix of elements, each of which is accessed by a random address code, allowing approximately equal access time to all bits. The simplest ROM structure is a network of diodes wherein the presence or absence of a diode determines the logic state of a particular location; such a network is shown in Fig. 11. The row address decoder raises the voltage on the appropriate word line to a high positive level, forward-biasing the diodes attached to that line. When the diodes begin conducting, they force their associated bit lines to a high (logic 1) level, while bit lines not connected to diodes remain low (logic 0). Output amplifiers sense the state of each line and present the bits to the data processor's other circuitry.

For example, if the row address decoder raises the word line 1 level, diodes D1, D2, and D3 conduct, raising bit lines 1, 2 and 4. In this case, the matrix output would be the binary number 1101. The next address might raise word line 4, in which case the output would be 0110. In some applications, column (bit line) addressing is added to select fewer than the maximum possible bit outputs.

ROM matrices are also formed with bipolar and MOS devices. In the most common configurations, the presence or absence of conductors establishes logic states.

Fig. 12 shows a ROM matrix utilizing multiple-emitter bipolar transistors. In this case, the collectors are used as row enabling contacts, replacing the word lines, and emitter contacts are omitted from selected locations to set logic levels. When the row address decoder raises the voltage on the appropriate collector to a

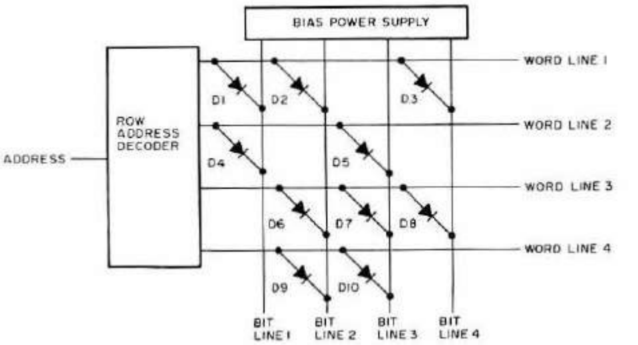


Fig. 11. A diode network with random access addressing is the simplest type of semiconductor ROM.

sufficiently high level, the transistor segments with emitter contacts begin conducting: for example, if Q2 is selected, the matrix output is 1001 (the level of columns 1 and 4 raised by conduction, 2 and 3 remaining low). In Fig. 12, column address and data output decoding selects two of the four bits for application to the processor.

In Fig. 13, a ROM matrix composed of static MOS FET devices is shown. Logic states are determined by the presence or absence of gates within the transistor structures. Reading this memory is accomplished in the same manner as diode and bipolar ROMs, except that the bit lines are driven low (to ground) when the FETs conduct.

Programmable ROMs

Semiconductor memories are almost universally formed on minute silicon chips capable of holding large numbers of integrated circuit devices; the chips often contain complete addressing, decoding, and output circuitry in addition to the memory cells.

In the formation of standard ROM matrices – in which the stored data is never to be changed – logic states are established during the manufacturing process by omitting the proper elements to create the desired bit pattern. This is the most prevalent type of read only memory. There are cases, however, in which standard memories are not available to meet application requirements, so programmable ROM (PROM) matrices are also produced.

A PROM is essentially a semiconductor matrix which has its program written into it at some time other than the manufacturing process. The manufacturer provides a chip on which all of the rows and columns (word and bit lines) are linked by conducting devices. Before integrating the chip into a circuit, the purchaser of the PROM uses various techniques – from application of a high-level write current to a laser beam – to eliminate devices from the matrix. In this way, a stored program unique to a given application can be produced.

New Developments

This discussion has covered structures that are representative of devices currently used in data processing memory facilities, and has not attempted to consider all of the variations of the basic structures. Advances are being made at a remarkable rate, and today's technology may be totally obsolete in a few years. Among the new memory devices that may bring this about are charge coupled devices (CCDs), bucket brigade devices (BBDs) and magnetic bubbles. Charge coupled and bucket brigade devices are similar in that both store digits as the presence or absence of electric charge.

A basic CCD is a semiconductor chip – either n- or p-type – over which a dielectric material is laid. A series of gate contacts are placed along the dielectric. Charge is stored as minority carriers under the gate regions. When the substrate is p-type, for example, applying a positive voltage to one gate attracts electrons out of the substrate until they dominate in the area directly beneath the gate, forming a "potential well." This storage condition is maintained for times up to several seconds after the gate voltage is reduced. Raising potential on the next gate in the series forms a second potential well into which the stored charge is transferred; gate potentials are sequentially raised by a clocked voltage to move the stored bit through the device. The CCD is thus a sequentially accessed memory facility similar to the shift register.

The movement of charge in a bucket brigade device is the same as in the CCD. Potential wells, however, are replaced by "buckets" of material unlike the substrate; for example, n-type areas may be embedded beneath the gates in a p-type chip to act as MOS storage capacitors.

Fabrication techniques currently limit the production of CCDs and BBDs, but they hold the promise of extremely small, very fast, low power memories of high density, and many manufacturers are investigating their commercial feasibility.

Magnetic bubble technology is still in the developmental stage, but it also shows great promise. The bubbles, which are tiny, mobile particles whose polarity is opposite to that of the thin film containing them, can be arranged to form coded data patterns, thus providing a storage medium.

Conventional bipolar and MOS devices are being modified to achieve an optimum combination of memory cell size, speed and power consumption. N-channel and p-channel MOS FETs are being combined on one chip as complementary MOS (CMOS) devices for low power applications, and Schottky diodes are being introduced into various bipolar configurations to decrease power consumption and increase speed. One such modification has resulted in the low power Schottky TTL memory cell, which has access speed approaching that of ECL (the fastest presently available cell type) and power requirements close to those of MOS FETs; GTE Lenkurt uses this family of devices in its 262A and 262B data sets to achieve the most rapid data processing possible with the least power. In another development, metal-nitride oxide semiconductors are being looked at as possible non-volatile read/write RAMs (memory facilities which

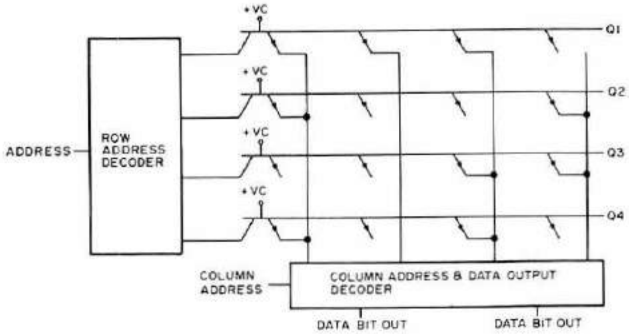


Fig. 12. Bipolar read only memory matrix.

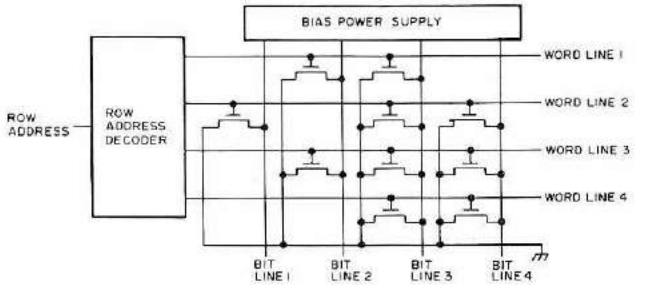


Fig. 13. The logic states within a static MOS RAM are determined by the presence or absence of gate contacts.

would not lose stored data when power is removed).

Whether improvements to existing structures continue at the present rate, or new technologies take over completely, there is no doubt that semiconductors will play an increasingly important role in data processing systems. ■

Bibliography

1. Altman, L. "Semiconductor Random-Access Memories," *Electronics,* Vol. 47, No. 12 (June 13, 1974), 108-110.

2. Carr, W. and J. Mize. ***MOS/LSI Design and Application.*** New York: McGraw-Hill Book Co., 1972.

3. Eimbinder, J., ed. *Semiconductor Memories.* New York: Wiley-Interscience, 1971.

4. Frankenberg, R. "Designer's Guide to: Semiconductor Memories – Part 2," ***EDN,*** Vol. 20, No. 15 (August 20, 1975), 58-65.

5. Kroeger, J. and B. Threewitt. "Review the Basics of MOS logic," *Electronic Design,* Vol. 22, No. 6 (March 15, 1974), 98-105.

6. Luecke, G. et. al. *Semiconductor Memory Design and Application.* New York: McGraw-Hill Book Co., 1973.

7. Orlicky, J. *The Successful Computer System.* New York. McGraw-Hill Book Co., 1969.

8. *Reference Data For Radio Engineers.* Howard W. Sams & Co., 1968.

I/O Editorial
From page 32

BASIC (A Self-Teaching Guide) takes a more sober approach and goes much farther. Programmed teaching, with step-by-step questioning of the student, punctuates the text into individual concepts, none of which intimidate the student the way a less cautious text can. After explaining input and output, arithmetic, and simple logical statements, the more sophisticated capabilities of BASIC, such as matrix arithmetic and string (character) operations, are cautiously introduced. After completing the text, the student should feel competent to handle most problems that can be solved with BASIC and a small computer.

No text in programming is really complete without a computer to practice on, and the versions of BASIC that different manufacturers supply are not all identical to any textbook. But no manufacturer's reference book takes the time to start a new programmer off on the right track without major confusion – unless perhaps he's a genius. For a timid user, *My Computer Likes Me* would be ideal as an appetizer, but *BASIC (A Self-Teaching Guide)* provides a main course for the hungry learner.

What to Do After You Hit Return (or, PCC's First Book of Computer Games). People's Computer Company, Menlo Park CA; 158 pp.

Computers are fantastic toys, and the best thing about them is that they can be made to do so many different things, so you never really get bored. Especially with this book, which includes fifty different games you can play on a computer that speaks BASIC. These were developed on the Hewlett-Packard 2000 minicomputer system, and the programs were all tested by H-P personnel. Many have been circulating for years; some are quite recent.

There are number games, word games, pattern games, board games, simulations, and even science fiction games. *Nim, Lunar Lander, Star Trek, Madlib, Bagels,* and most other popular computer games are included. Rules are explained, sample runs are shown, and in most cases, program listings are included (in BASIC). Most can be played without a computer if you're so inclined, but the hardware is half the fun, right?

What to Do After You Hit Return is big and attractively put together on giant pages that stay open on the table. It's the software equivalent of a barrel of monkeys. See why most colleges find that most of their computer time is spent playing games!

I/O EDITORIAL

WANT TO MAKE A BUNDLE?

Not to be a name dropper, but sure as God made Yellow Transparent apples (to me the finest apples in the world) someone may make a fortune by coming up with a small computer system for small business and home use . . . if he plays his cards right. Best of all, nothing further has to be invented to make this a reality . . . some of the pieces have only to be put together.

Since a small black and white television set retails for around $85, a keyboard for $30, a microcomputer chip for $20, a cassette recorder for $20, how much could a small computer system cost embodying these elements? Oh, you'd need a display generator for the TV, an interface for the cassette, and some working memory. I'll bet someone could put such a contraption on the market for under $1000 in six months . . . and the price would be down to half that in six more . . . and eventually down to maybe $250 . . . or $125. The prices go down a lot when chips come available to handle each function . . . and eventually the whole works.

They're already starting to put RAM memory on CPU chips . . . and I/O interfaces too. Add a small built-in ROM operating system and a TV display generator . . . and the price plummets for a complete small computer system.

Do you want to wait for someone else to get into the business or start working on it yourself? Here's a way for someone to invest some time and money in a project which could be worth millions in a couple of years. MITS had about 15 people when they entered the uP business . . . now look at 'em!

HAM COMPUTING

Most of us can think of a lot of good things we might do if we had a computer . . . like keep track of stations contacted along with some details about them . . . maintaining an index to ham magazine articles that we might want to look up . . . determining Oscar intercept times and bearings . . . running all the functions of a repeater . . . playing any of the hundred or so popular computer games . . . getting into computer art forms with a color television set . . . experimenting with computer music . . . keeping track of repeater channels and locations . . . Morse code conversion . . . RTTY operation . . . things like that.

But the next question is a tough one . . . what hardware will it take to get involved in my interest . . . and, even more difficult, what programs . . . and where can I either get the programs or get the training so I can develop my own programs?

Frankly, we could use a lot of input on both hard and software. Let's see some articles on this . . . showing configurations of available gear which will do ham and hobby jobs.

If we look at the Sphere equipment we see that for most ham applications we will need their "200" system, their "B" package and BASIC on cassette. This comes to $860 for the "200," $205 for the extra 4K RAM memory and character generator ROM of package "B" and $100 for BASIC on a cassette . . . total $1165 for the hardware. Now, if you buy the BASIC textbook by Albrecht ($4), you will be ready to go with almost any program you want, doing your own programming. The DEC 101 Games in BASIC ($7) will launch you into the game biz. You will need a television set for the display.

The Sphere system will give you the computer, a character generator so you can read out the input and output of the system on a television set, a keyboard for inputting, BASIC programming language which takes about 5K of RAM memory, leaving about 3K for your use, and an I/O for a cassette recorder for use in entering programs, storing programs for later use, and storing data in long-term memory. That's a fairly complete small system.

If you want to go the MITS route you'll need an Altair 680 ($420), 12K of RAM memory ($825), an I/O port to interface the 680 with a video display terminal ($144), BASIC language on cassette ($75), cassette interface I/O port ($144), plus some sort of video generator and keyboard unit such as the Southwest Tech kit which runs about $282 (and then you have to build it . . . which is fun). The cost of this system would appear to come to around $1890. If you want to go the Altair 8800 route . . . and Ed Roberts points out in the Altair Computernotes publication that the 8080 chip is substantially better in his estimation than the 6800 chip . . . this would increase the cost even more.

The Altair 8800 computer is $621, expander boards are $93, a cooling fan is $20, three 4K RAM boards are $825, the cassette I/O is $174, the television typewriter I/O is $144 and BASIC language on cassette is $75, plus the $282 for the TVT kit comes to $2234.

To do any work involving data you want to keep on file you will need a couple more cassette control systems so you can use one cassette for your data base, a second for any update to the data base, and a third for the

Continued on page 47

John A. Lehman WB8TUR
716 Hutchins #2
Ann Arbor MI 48103

Computer Languages-- Simplified

Anyone who wants to use a computer has to have a way to communicate with it. This article is a simple introduction to some of the languages which are used for that purpose. It is intended for rank beginners, so all of the programmers, software freaks and computer hot dogs in the audience might as well stop here. For anyone else, I'm going to try to keep everything in English (which is not a computer language, unfortunately) and avoid as much computerese as possible. So here goes.

One might start by asking, "Why have computer languages at all?" Back in the dark ages 25 or 30 years ago they didn't – the machines were wired up to do a certain thing and that's what they did. But, somewhere along the road, some bright fellow realized that it would be much more efficient if you could feed the machine a fairly long set of instructions and let it follow them. This also made for much greater flexibility, since you could give the machine different sets of instructions. These instructions are what a computer language communicates, and this article will go over some of the more common ones, to wit: Assembler, BASIC, FORTRAN, PL/1, COBOL and a little bit about some of the more specialized ones. But first let's look a little bit more at the nature of the beast we're dealing with.

It is helpful to think of a computer as a glorified electronic calculator. In fact, some of the more modern calculators really are computers. But let's look at the average four function calculator. It has a display, which is called an output device in computerese, and it has a keyboard, which is an input device. To do anything with it, you have to enter the numbers through the keyboard and then enter what you want done with them, be it to add them or whatever.

Machine language is the closest we can get to what the machine actually speaks.

But suppose you want to do a mortgage calculation – say, figure out what your interest payments and principal payments are going to be each month for the life of the 20 year mortgage. This means that you're going to have to do a very repetitive calculation 240 times. It is to avoid this sort of hassle that real computers (and some fancy calculators) have stored program capacity – a stored program being nothing but a set of instructions inside the machine which get done without your standing there pushing all of the buttons each time. But now we have to get this set of instructions inside the machine, and that is where programming languages come in. We need a way to communicate the instructions to the machine.

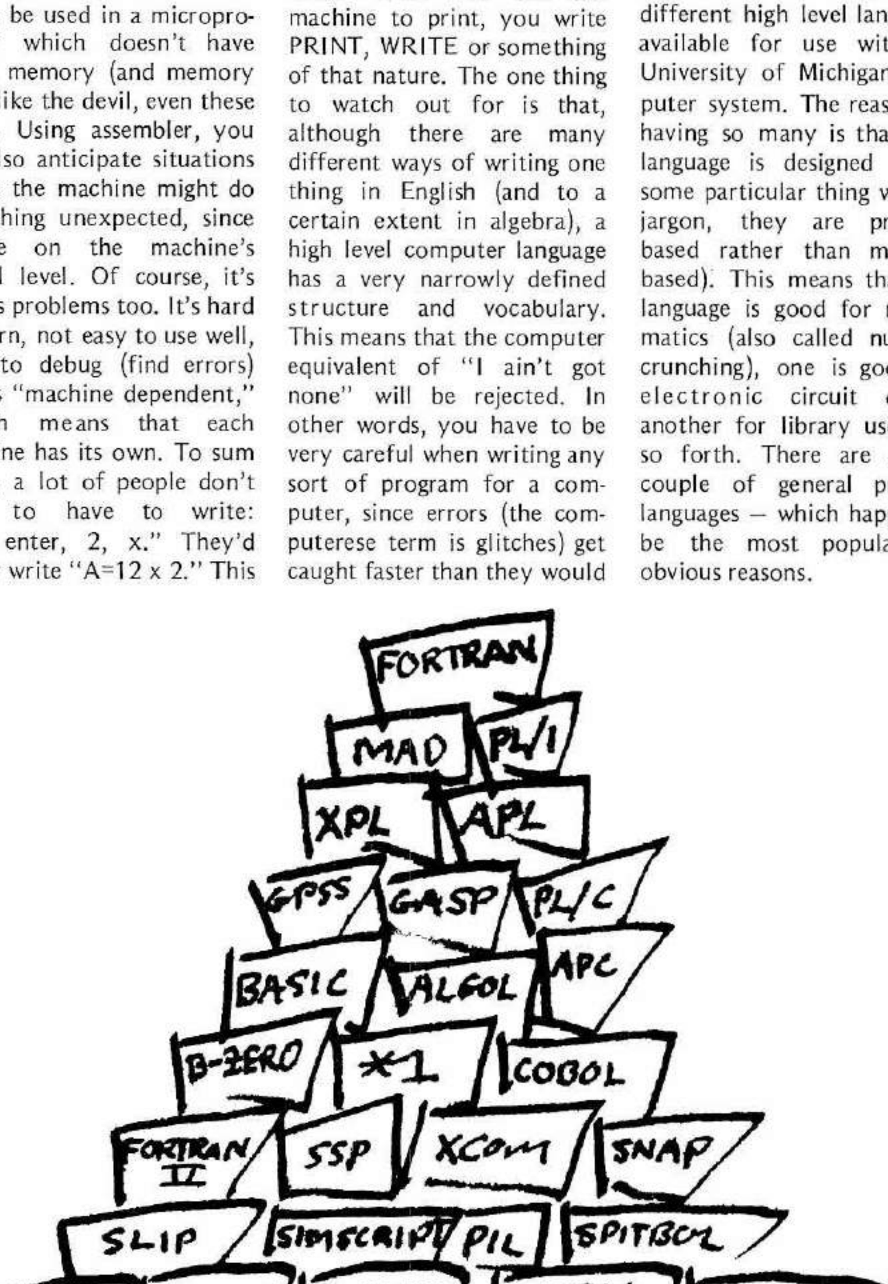
In the case of a calculator, this isn't much of a problem. The "Instruction Set" (list of all of the instructions which the machine is able to follow) is hard-wired in. If you want it to add, you push +. This is obviously not too good an idea for a large computer, since the number of buttons gets pretty large. Besides, you tend to run out of symbols, which makes everything even more confusing. So we need some sort of language. The simplest one is called "machine language" and is the closest we can get to what the machine actually speaks. However, as any of you who have been following the articles in *73* about gates and such know, computers and other digital machines run on high and low levels of voltage. Since this is rather hard to see, we usually represent it with ones and zeros, or on and off lights. The whole thing is like trying to communicate using RTTY but doing it by ear instead of with a TTY machine, which is to say that it's a royal pain. Anyone who's into interpreting things like 01001101 01100001 and so forth can really get off on it, but for most of us there's gotta be a better way. Fortunately there is. Incidentally, if you ever get a microcomputer, those switches and lights on the front panel are used to communicate with the thing in machine language.

The next sort of language developed, and the one which is most widely available for microprocessors these days, is the assembler. Each type of computer has its own version; what it is, in short, is machine level logic – but using symbols and normal numbers rather than ones and zilches. Assembler is related to what you do with a calculator – in fact, any of you who own or use HP calculators have been using a version of assembler language usually known as Reverse Polish Notation. With an assembler language, you specify what number you want, where you want it put and what you want done with it; for example (to use HP assembler): "12, ENTER [*which puts it in the region where the arithmetic is done*], 2, x" multiplies 12 times 2 and comes up with

24. All assembler languages work this way, although many of them have dozens of commands and hundreds of locations where things can be put or obtained. This sort of computer language has a lot of advantages. It's very efficient not only where memory is concerned, but also with regard to execution time. This means that it's cheap to use. The assembler (the program which translates it into machine language) doesn't take up much memory either, which means it can be used in a microprocessor which doesn't have much memory (and memory costs like the devil, even these days). Using assembler, you can also anticipate situations where the machine might do something unexpected, since you're on the machine's logical level. Of course, it's got its problems too. It's hard to learn, not easy to use well, hard to debug (find errors) and is "machine dependent," which means that each machine has its own. To sum it up, a lot of people don't like to have to write: "12, enter, 2, x." They'd rather write "A=12 x 2." This is what "high level" languages let you do, along with all sorts of other convenient things. For this reason, almost all programming these days is done with one of the various high level languages, and the rest of this article will be about some of the more common of them.

First of all, a high level language is a computer language that is based on some combination of English and algebra. So, to write two plus two you would usually write "2+2." To tell the machine to print, you write PRINT, WRITE or something of that nature. The one thing to watch out for is that, although there are many different ways of writing one thing in English (and to a certain extent in algebra), a high level computer language has a very narrowly defined structure and vocabulary. This means that the computer equivalent of "I ain't got none" will be rejected. In other words, you have to be very careful when writing any sort of program for a computer, since errors (the computerese term is glitches) get caught faster than they would be by an old-fashioned high school English teacher.

Continuing the comparison with human languages, there are lots of different ones for computers, too. At first each company developed its own; now many of them are standard and thus can be used on any machine with few, if any, changes — unlike assembler languages. We still have a lot of computer languages, though. For example, the last time I checked the documentation, there were something like 35 different high level languages available for use with the University of Michigan computer system. The reason for having so many is that each language is designed to do some particular thing well (in jargon, they are problem based rather than machine based). This means that one language is good for mathematics (also called number-crunching), one is good for electronic circuit design, another for library use, and so forth. There are also a couple of general purpose languages — which happen to be the most popular for obvious reasons.



It would be a pain to have to spend a few weeks mastering a new computer language every time you wanted to do something different with a computer — not to mention that you have to buy (or write) a special translating program (called a compiler) for each one — and that can get very expensive (much more than the cost of the computer itself). So let's look at some useful, fairly general purpose languages.

BASIC, which stands for "Beginners' All-purpose Symbolic Instruction Code," was developed at Dartmouth College a number of years back. It was designed for people who knew nothing about computers but who wanted to use them. Few dyed-in-the-wool programmers care much for it, but most non-programmers love it. It's based on algebra, and the non-algebra parts of it are in plain English. For example, to enter two numbers into the machine, multiply them, divide one by the other and print the results, you would write:

```
1 IN A, B
2 LET C = A * B
3 LET D = A / B
4 PRINT C, D
```

(* is the standard symbol for "times" on a computer)

This language has quite a few advantages. It's easy to learn, easy to use, and there are lots of books around which help people learn it. Equally important, there are lots of programs already written and published in it (these are called canned programs), and quite a few computers can use it. In particular, Altair has two (going on three) versions out at the moment, and other microcomputer manufacturers are making noises about supplying it — or so I read. The compiler (remember, that's the program which translates the things you write into the machine's language) doesn't take up too much memory either, which means that BASIC is suitable for small computer systems where memory is limited. A final advantage is that BASIC is fairly flexible – especially the advanced systems. You can do many if not most of the same things with it as you could do with FORTRAN or PL/1, although the programming effort might be greater.

It does have some disadvantages, though. BASIC has no mnemonic variables. This means that you have to remember that A stands for current, E for voltage and so on. In a more advanced language you could write AMPS, EMF, etc. This isn't so bad if you're working with standardized symbols, but gets to be a disadvantage when you try to remember which was Accounts Payable and which was Accounts Receivable. Also, BASIC is somewhat limited as to what you can do with Input/Output. This only makes a difference if you are working with a big system that gives you lots of choices – it isn't of too much concern for a home computer system or a small business one. Finally, BASIC is structured somewhat along the same lines as FORTRAN, which is the oldest computer language still in use. This means that it does a lot of things in harder more roundabout ways than some of the newer languages, like PL/1. For example, its "either-or" choice is rather cumbersome to write. It's still a great language to play around with, though.

FORTRAN is probably the best known of the various computer languages, partly because it's one of the oldest. The name stands for FORmula TRANslation, and it was developed by IBM back in the early 1950s. It and the B-Zero language developed by Univac were the first high level languages used. No one uses B-Zero today (few have even heard of it), but FORTRAN is probably the most widely used computer language in the United States. Of course, the FORTRAN we use now isn't the same as the FORTRAN introduced back in 1957 – just as the English we speak now isn't the same language as the people in England spoke back in 1066. There have been three official versions of FORTRAN: FORTRAN (the original), FORTRAN II and FORTRAN IV. Number three got lost in the middle somewhere. Most computers these days use FORTRAN IV, although there are some minicomputers around that still use FORTRAN II; some of these compilers might be adaptable to microcomputer use. Anyway, as you might guess from the name of the beast, FORTRAN is basically a scientific computer language; it was developed to make it easier to solve mathematical-type problems for science and engineering. Over the years the language has expanded to the point where it is usable as a general purpose language, so it can do a lot more than simply crunch numbers. Moreover, since it's such a popular language, there are who-knows-how-many programs written (and sometimes published) in it, which makes it much easier to solve a given problem (since you can frequently just type in a canned program). The same structural problems encountered in BASIC are part of FORTRAN, but these have already been covered. Perhaps more important for anyone who wanted to use FORTRAN on a microcomputer, the compiler (remember? the program which translates it into machine language) takes up much more memory than a BASIC compiler, though much less than one for most other high level languages. One keeps hearing hints that one of these days someone may develop a version which is usable on a microcomputer, but I haven't seen any announcements yet.

While FORTRAN was developed for scientific use, COBOL was developed for business use. It's the language used by the U.S. government for a lot of their stuff, so it's got a pretty wide circulation. Needless to say, many businesses use it, too. From the little work I've done with it myself, it seems that you spend most of your time defining what your printout is going to look like and what the information which you feed into the thing is going to look like (the jargon for this is format definition). It also takes lots more memory than one would probably want to pay for in a microcomputer – unless he wanted to do sub-contracting for the government.

BASIC . . . was designed for people who know nothing about computers, but who want to use them.

Up until the early 1960s, most computers were either scientific- or business-oriented, and a machine which was designed for one didn't usually work too well for the other. Then IBM started making their general purpose machines, and most other people followed suit. At about the same time, people started to worry about a general purpose computer language. A number were developed, of which my favorite (just for the name) is MAD (Michigan Algorithmic Decoder) which was brought to us by the folks at the University of Michigan Computing Center. Then IBM got into the act, and, lo and behold, out popped PL/1 (Programming Language One). The best description that I can think of is that it was designed to out-fortran FORTRAN and to out-cobol COBOL all at the same time. It does a pretty good job of it, too. I'm always amazed at all of the nice things you can do with PL/1; to use the computerese phrase, it has more bells and whistles (extra options) than you can shake a stick at. Unfortunately, it also uses more memory than you can shake a stick at, which makes it too expensive for microcomputer use (or even timesharing use if you have to watch your costs). But never fear, one of these days we may be seeing a scaled down version of PL/1 (called PL/M) which keeps a lot of the nice features without taking up more memory than most of us mere taxpayers can afford. To give an example of the sort of nice thing the language can do (among others), it lets you write a simple either/or statement (if this is true, do one thing; otherwise do this other thing), whereas to do that in most other languages you have to play hopscotch with the line numbers. In short, PL/1 is a great language, and if a cheap compiler ever comes out, I hope I own the microcomputer it's written for!

Another new language, again by IBM, is APL. I will confess here and now that I've never used it, so what I say is taken from what people who have used it have told me. This is a very powerful language; it can do in one line what most other languages require five or more to do. It is not yet widely used; unless I'm mistaken (which is quite possible) IBM is the only company which makes compilers for it. It *is*, however, designed for the sort of sit-down-at-your-computer use that most hobby users probably have in mind. And good news! It's available for a microcomputer. The bad news is that said machine is the IBM 5100 and it runs about 10k — and those are kilobucks, not kilobytes. Anyway, the scuttlebutt has it that APL is one of the languages to watch, so keep your eyes peeled.

I mentioned a while back that, in addition to the general purpose languages I've discussed so far, there are quite a few special purpose ones. For a hobby user (or would-be user) these aren't too important, but just to be more or less complete I'll mention a few which are good with which to impress people (besides being good to know about if you tend to be around computer hot dogs who like to talk about such things). There is RPG, which is a Report Generating language, and therefore much used for business and that sort of computing. Then there are several languages used to write simulations. (A simulation is like a computer game except people take it seriously. Come to think of it, I know a couple of people who take the Star Trek game seriously, but we'll ignore that.) These languages are things like GASP, GPSS and so forth. There are a fair number of languages used for crunching words instead of numbers — SNOBOL, SPITBOL and so forth. Finally, there is at least one language used to design electronic circuits — unfortunately all my electrical engineering friends seem to be able to tell me is that it exists and Professor Zilch mentioned it.

As a brief review, one needs some sort of language to give a computer instructions. You can operate on the machine's level (called machine language) and feed it ones and zeros. Or, you can stay on the machine's logical level but use decimal numbers and abbreviated commands. This is called an assembler language, and requires a separate program for translation into machine language. Finally, you can use a combination of human logic and normal words and symbols, which is called a high level language. This requires a program to translate it into machine language again, and such a program is called a compiler. The principal advantages of high level languages are that they are easy to learn, easy to use, and are the same for any machine. They are not nearly so efficient as assembler language from the computer's point of view, but they are much more efficient from our point of view — and that's usually what counts. ■

FORTRAN ... does things in harder more roundabout ways than some of the newer languages, like PL/1.

References

1. ***BASIC,*** **Albrecht, Finkel and Brown. Wiley & Sons, 1973.**
2. ***Digital Computing, FORTRAN IV, WATFIV, and MTS,*** **Carnahan and Wilkes. Chemical Engineering Dept., University of Michigan, 1973.**
3. ***High Speed Data Processing,*** **Gotlieb & Hume. McGraw-Hill, 1958.**

I/O Editorial From page 43

updated data base. This means two more I/O interfaces and control systems ... it also means a lot of programming using BASIC. If you want to keep a file of every station you've contacted and be able to quickly get the data on someone, this is the system you'll probably want.

The same system would work fine for keeping track of magazine articles you might want to reference, recipes for the wife, addresses for a club, a file on music in your record collection, a list of repeaters, etc.

These systems will be a little slow in locating data in memory since they will have to scan your cassette tape for the data file wanted. It won't be long before we have relatively inexpensive tape systems which can be searched much faster ... and relatively low cost floppy disk memory systems are on the way. Disks don't hold much more than cassettes, but they search very rapidly.

There are a lot more small computer systems coming along, but I haven't got much data on them as yet. It is almost impossible to understand some of the advertising literature in this field and even talking with the manufacturers can be frustrating, for some of them are so busy designing and turning out parts of their systems that they haven't given much thought to what it will be used for ... or what problems the user may run into.

I would like to state that *73* is most anxious to hear from any ham computer hobbyists who have managed to put small systems together and get them to actually do something. Please let us know what you are using and how you programmed it. I would also be delighted to hear from *any* manufacturer who can state just what hardware and software is needed ... and how much it will cost ... to set up some hobby systems such as outlined above.

For instance, any of the small systems being marketed are quite capable of calculating the Oscar acquisition times, but where can you get the program for the calculation? Unless it is in either machine language for your particular chip or in BASIC, you won't be able to do much. And it has to be in a BASIC that is compatible with the BASIC you've got for your computer, just to make matters a little more complicated, for there are BASICS and then there are BASICS, from Mini-BASIC on up through Extended BASIC. Obviously we all need to know a lot more about this situation ... and get some counsel on where to get the programs we need for Oscar, moonbounce times, etc.

Now is the day of the true pioneer as far as microcomputers are concerned ... like sideband in the mid-50s ... like FM in 1969 ... and RTTY in the late 40s. We are a long way from the appliance operator phase of uP systems and these are the days which separate the men from the boys. Let me emphasize again that having fun with computers does not take super brains ... it does not take a whole lot of money ... it does not take education ... all it takes is enthusiasm and persistence. And, if you are young in spirit, there is a very good chance that you can parlay your knowledge into a very comfortable living.

THE MAGIC OF FRESH ORANGE JUICE

The other morning, as I was squeezing oranges, I had a depressing thought. I remembered a bit on a recent Today show lauding the wonders of Florida. It was a film about the citrus industry there and they mentioned that two thirds of the Florida oranges are now processed into frozen concentrate. How many years since you have had fresh orange juice for breakfast?

Once you get into fresh orange juice you have a tough time gagging down the frozen stuff ... or the "fresh" juice in cartons. You have to totally forget what orange juice tastes like to accept frozen. An electric juicer runs around $10 to $12.50 (Unity or Sears) and you can crank out a glass in about a minute ... you don't thaw out a can of frozen much quicker.

Just as we have all come to accept the frozen juice alternative to do-it-yourself juice, with a loss of substance, most of us hams have come to accept our ham gear as either ready-built or in kits (paint by the numbers?). I have a strong feeling that the computer revolution in amateur radio will bring back a lot of the home squeezed flavor to our hobby ... for, even if you work from an assembled or kit computer, you are still only about 25% of the way toward your goal. This is one field where the hardware is only the start and merely opens the gates for self-expression and creative fun via the programming you will be doing.

I hope that the manufacturers of computers won't be angry at me for letting the cat out of the bag, but the fact is that getting your equipment working is only a small part of the fun and challenge. Oh, in time a lot of the work will be done for you and will be available on .tape or something ... perhaps ROMs. At least, programs will be available for ordinary uses of the equipment ... you will still be on

Continued on page 51

73 Magazine *Staff*

A Nifty Cassette - Computer System

The most practical and economical way to store programs and large quantities of data for small computer systems is with the common tape cassette recorder. Cheap and plentiful, audio-type cassette equipment is capable of storing several times the amount of data that an equivalent volume of paper tape can hold, with the added benefits of erasability and easier operation. Floppy disks may be faster, but are beyond the price range of most hobbyists.

While computer manufacturers and software houses have long been supplying their programs on audio cassettes, there has been a major problem with compatibility. Every manufacturer has had his own pet system of recording, and a tape recorded for use with one brand of computer is utter gibberish to another brand of computer. For this reason, several manufacturers decided to adopt a standard system of tape interfacing.

The proposed standard, as implemented by Pronetics Corp., calls for a frequency-shift keying standard not entirely dissimilar to that used for RTTY, but with several crucial differences. The two tones to be recorded are ideally to be square waves, with Mark (logic 1) to be 2400 Hz, and Space (logic 0) to be 1200 Hz. With a standard tape exchange speed of 300 baud (bits per second), Mark would consist of eight cycles, Space of four. This could be divided to 600 or 1200 baud, in which case one cycle would be a space (1/1200 sec). Higher density would be impractical. For comparison, 300 baud corresponds roughly to 30 characters per second.

Within each character, the first recorded tone should consist of a Space (start) bit, followed by eight data bits (least significant bit first, parity last) and two Mark (stop) bits. All undefined bits, as well as the interval between characters, would be Mark (2400 Hz).

This system has several beneficial features. It is self-clocking. The first bit of any character is Space and must follow the Mark tone that ends previous characters and exists between characters. It is possible to tolerate as much as a 30% speed variation with this system, which can be an important factor with inexpensive tape equipment.

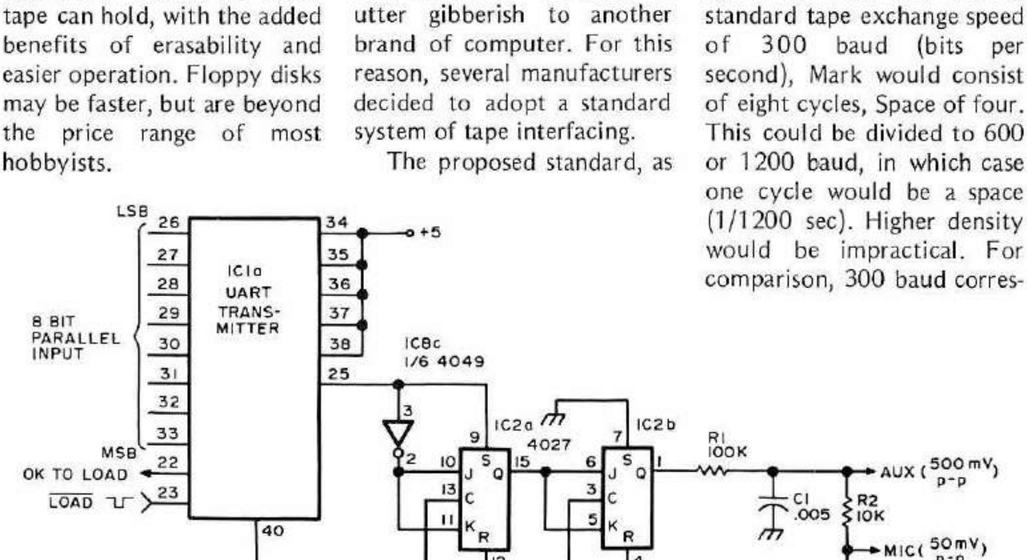


Fig. 1. Cassette digital modulator. This circuit converts 8-bit parallel input data to a series of 2400 and 1200 Hz tones for recording on cassette tape.

Do I Need a Good Recorder?

Almost any cassette recorder can be used for data storage using this FSK standard system. But for convenience and accuracy, there are a few criteria for selection that differ from hi-fi quality.

You want a clean, reliable machine. Dirty heads and mechanism can spoil data very easily. If you don't have a cassette recorder already, a used model is adequate, but it shouldn't show signs of mistreatment. It should also have capstan drive – a few miniature units don't.

A digital tape counter is also a great convenience. Without one, identifying programs on a tape can be difficult, and could lead to accidental erasures. These are found on many hi-fi type machines, and occasionally on portable units.

An important electrical feature is ac bias/erase. Some recorders, and most of the under-$100 category, use dc for erasing and record biasing. This results in higher noise and less frequency response. While frequency response is not as critical with this system as with music recording, it still helps to have a good clean treble response, which can help preserve the square wave shape. Low noise means fewer errors, so a high signal-to-noise ratio aids reliability.

Stereophonic capability is unnecessary. If you have a stereo recorder, be sure to record both tracks simultaneously, and bulk erase the tape before using it.

Your tape recorder must have an auxiliary or microphone input jack, as well as an earphone or line output. Acoustic coupling is unsatisfactory. The choice of levels can be performed in the computer interface circuitry, so either mike, line, or speaker levels can be used.

In a meeting organized by the staff of 73 Magazine, the microcomputer industry accepted this cassette system as the industry standard.

What About Tape?

While the choice of tape recorder is uncritical, the tape itself is the weak link in the chain. Do not skimp on tape. Use the best tape you can get your paws on. Since dropout on the tape means loss of data, the tape must have a high manufacturing standard. Some cassettes jam easily, and the thin tape found in C90 and C120 cassettes is too thin and fragile to be reliable. A premium grade C60 tape is ideal. Perfectionists might want to spend the extra money for chromium dioxide tape. The extra response can't hurt.

Store the tapes in a dust-free location, in their own container. Do not smoke near the tapes or the recorder, and clean the heads frequently. Tape cleanliness and quality are far more important in digital applications than in music.

The Recording Interface

Digital information from your computer is generally available as 8 bits parallel from either an I/O port or data bus. The tape is recorded serially; the conversion is best accomplished with a Universal Asynchronous Receiver/Transmitter (UART) IC.

The modulator is shown in Fig. 1. The serial output of the UART has logic 1 as a high level and logic 0 as a low level. IC2a and IC2b form a clock divider circuit, dividing the 4800 Hz clock signal by 2 or 4, depending on the UART output level. The output is a series of square waves which feed the tape recorder's input.

The poor frequency response of some tape recorders, especially those with dc bias, causes the manufacturers to exaggerate the treble being recorded, which distorts the square wave. Sine waves record better, but are harder to generate digitally. In some cases using a low pass filter makes the waveform usable; R1 and C1 perform this function. A smaller value for C1 may increase effectiveness with better recorders. Fig. 2 shows the effect of the recording process on digital waveforms.

The AUX output of the interface is 500 mV peak-to-peak and is for use with high impedance high level inputs. The MIC output is 50 mV, suitable for most units with microphone inputs.

The 4800 Hz signal must be capable of driving two TTL loads. While a crystal oscillator and divider chain work best, and a phase locked loop referencing the 60 Hz power line is also very good, the oscillator in Fig. 3 is simple and quite satisfactory (but requires calibration with a frequency counter).

If the available digital information from the computer is already in serial form with the necessary start and two stop bits, and is properly timed at 300 baud, the UART is not necessary. However, the 4800 Hz clocking signal should be synchronous with the serial data, with 16 clock pulses per bit. If the serial data is not at 300 baud, a UART receiver must first be used to convert the data to parallel form. It then is clocked through the UART transmitter as shown.

The OK TO LOAD line on the UART goes high when it is ready to accept a byte of parallel data. The data is then loaded into the UART transmitter by pulsing the LOAD line low for at least one usec or until the OK TO LOAD line goes low. The transmitter will then start transmitting the byte when the LOAD line is returned to the high state. When not transmitting, the output is high, causing the modulator to generate the 2400 Hz Mark signal.

The Playback Interface

There are several possible ways to recover the FSK signal from the tape. An FM discriminator or a phase locked loop demodulator can be used, just as with an amateur RTTY signal. Users of previous nonstandardized cassette interfaces can readjust them to decode the 1200/2400 Hz tones, but the most accurate system uses

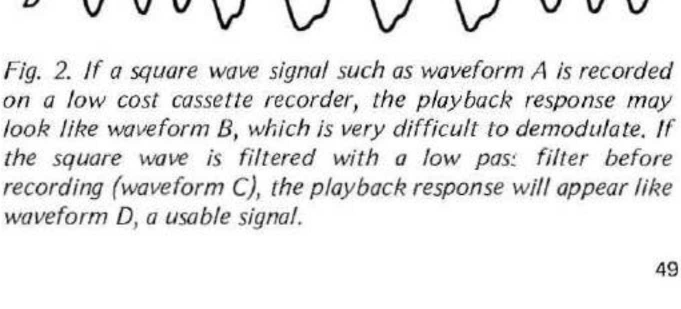


Fig. 2. If a square wave signal such as waveform A is recorded on a low cost cassette recorder, the playback response may look like waveform B, which is very difficult to demodulate. If the square wave is filtered with a low pas: filter before recording (waveform C), the playback response will appear like waveform D, a usable signal.

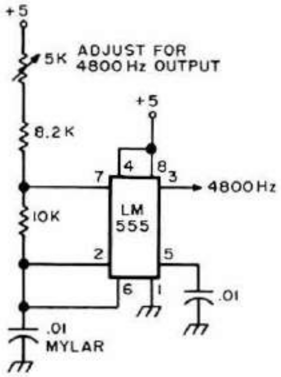


Fig. 3. Circuit of 4800 Hz oscillator. Use this circuit if a more precise and stable source of 4800 Hz is not available.

digital recovery to extract timing information from the recorded signal and uses that information to retime the recovered data.

Fig. 4 is a complete schematic of the playback demodulator. The signal from the cassette player is conditioned by IC3, an op amp used as a Schmitt trigger. The output of a Schmitt trigger is, by definition, either fully high or low, so it regenerates pure square waves from the distorted tape input. IC4 is a retriggerable one shot with a period set to 555 microseconds. As long as the input signal is 2400 Hz, the one shot is retriggered before it times out. Flip flop IC5a remains high, which is interpreted as logic 1. The 1200 Hz signal, on the other hand, has a period between pulses of greater than 555 usec, so the one shot times out, resetting IC5a. It stays at logic 0 as long as 1200 Hz is being received because the one shot is timed out whenever the next triggering edge occurs. When the 2400 Hz signal returns, the one shot stays high, permitting IC5a to switch back to high state. The output of this flip flop is the serial data.

While that simple circuit will work well if the tape speed is accurate to better than ±6%, such is frequently not the case. Since tape speed variations will be reflected in pitch variations in the recovered tones, it is possible to use the 1200 and 2400 Hz signals from the tape to retime the recovered data. Flip flops IC6a and IC6b extract this timing information. When the 1200 Hz signal is received, IC6a is preset with a pulse generated by C8 and R15 every time the one shot times out. The effect is to cause IC6 to divide by two. When 2400 Hz is being received, the one shot does not time out and IC6 divides by four. The result is a clock at the output of IC6b, at 600 Hz.

Instead of clocking the data into a shift register, the receiver portion of UART IC1 is used. It has built-in circuitry to identify the start and stop of each byte automatically. It also has three-state output (logic low, logic high, and functionally disconnected), which permits direct connection to most data buses and I/O ports. The UART needs a 16x clock, which is formed by phase locking a 4800 Hz oscillator to the 600 Hz output of IC6b. The PLL is adjusted to oscillate at 4800 Hz in the absence of any input signal. IC5b and IC9 divide the PLL output by 8 to drive one of the phase detector inputs, while the other input is driven by IC6b.

The UART receiver raises its DATA AVAILABLE output to logic 1 when it recognizes that it has received a complete character. Since the UART outputs are three-state, it is necessary to drive the RECEIVED DATA ENABLE input to logic 0 to read the parallel output data. After the parallel data has been read it is necessary to pulse the RESET DATA AVAILABLE line to prepare the UART to output the next byte. The pulse must remain at logic 0 for at least one usec, or until the DATA AVAILABLE line drops to logic 0.

Circuit Adjustments

The only adjustment necessary for the recording modulator is to put the 4800 Hz signal exactly on frequency. Since a Mark byte

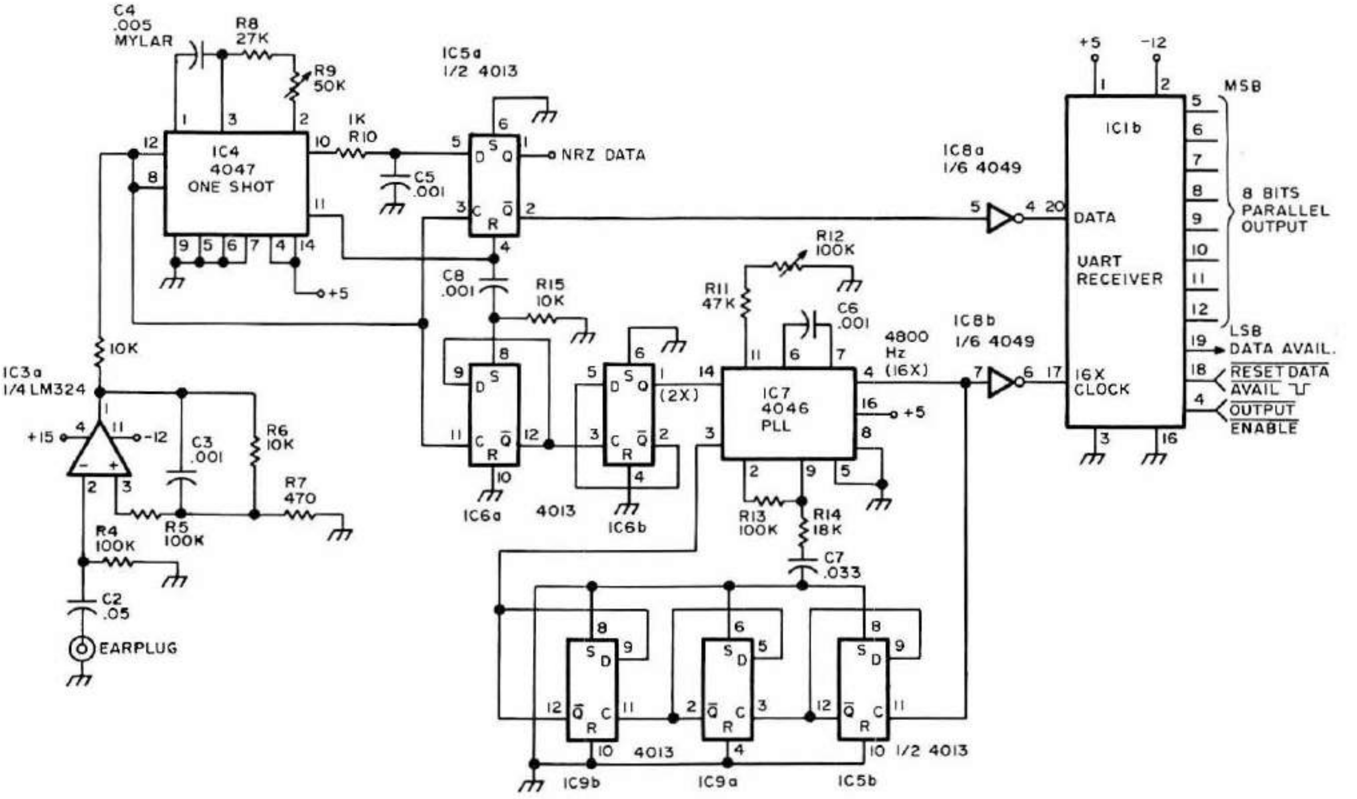


Fig. 4. Cassette data recovery circuit. Refer to text for circuit operation.

consists of eight (not seven or nine) cycles at 2400 Hz, this is fairly critical.

The data recovery one shot and PLL oscillator must be accurately adjusted for best results. The one shot is critical. To adjust it, set a well calibrated audio source to 1800 Hz with 1.5 to 3.5 V rms output. Adjust R9 until the data output of IC5a pin 1 just changes, measured on a high impedance voltmeter. Adjust R9 to as close to the point of change as possible.

The PLL oscillator is adjusted by R12 with no input to the playback input. If no counter is available, the oscillator output at IC7 pin 4 should be compared to the 4800 Hz signal used for the UART transmitter.

Circuit Operation

The circuit as shown will recover data most accurately if the earplug output signal of the tape recorder is between 4 and 10 volts peak-to-peak. Most portable recorders have that capability. If the cassette deck does not have a speaker amplifier, a low gain amplifier may be necessary. If the recorder uses dc bias, there may be too much treble, which necessitates turning down the tone control.

To comply with the standard for tape exchange, the recorded data should be preceded by at least 5 seconds of 2400 Hz tone before the data begins. This is accomplished by operating the recorder in the record mode for five seconds or longer before sending data to the UART transmitter. With the UART idle, the modulator generates 2400 Hz.

During playback, wait a couple of seconds before allowing the computer to accept the UART receiver output, to avoid reading the garbage generated by turning the recorder on and off. It is possible to have the computer control, via a relay, the remote control switch of the tape recorder under program control. It is still necessary to wait a few seconds before accepting data, due to the time spent starting and stopping the tape. The 2400 Hz leader provides that interval on the tape.

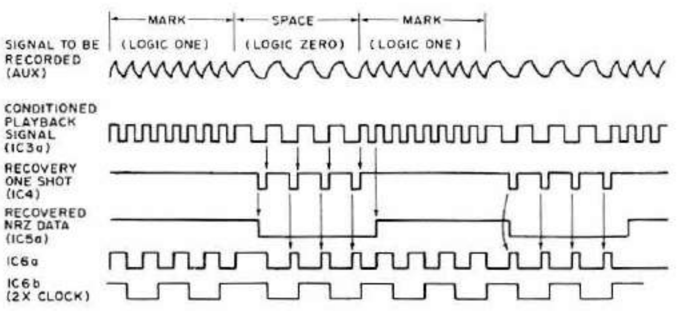


Fig. 5. Cassette modulator/demodulator waveforms.

Conclusion

Using this type of hardware for tape cassette modulation and demodulation simplifies programming for a cassette-oriented computer system. In some circumstances it may be possible to connect the interface hardware directly to the computer, while some computers may require peripheral interface adaptors to get the data in and out of the computer. ■

The cassette interface described here is manufactured by Pronetics Corporation. It is available fully assembled and tested on a 4.5" x 6.5" circuit card with standard edge connector. For price and other information write: Pronetics Corporation, PO Box 28582, Dallas TX 75228. – Ed.

I/O Editorial From page 47

your own if you want to customize the programs to fit your exact needs or if you want to use the gear for some new application. Who knows, once you're set up you might want to score a ham contest and program your computer to cross check every reported contact on the incoming logs . . . that program would keep you out of trouble for a while. Or you might want to analyze Oscar contacts for anomalies to see if there are indications of over the horizon propagation at times.

The uses for your computer will be expanding constantly as you discover new things you want to do . . . and learn more about how to program the beast to do what you want. When you have worked several weeks in your spare time to perfect a program, you are on the one hand a bit reluctant to just give away all that work to anyone who comes along . . . on the other hand you are so proud of what you have accomplished that you want others to see how neat it is. I don't know how this will work out in the long run . . . perhaps there will be some programming sales services to help you get a small royalty for your effort . . . selling your program to those who want it for a few bucks. On the other hand, the magazines of the future may devote pages to new programs . . . we'll see.

Right now you can buy some programs on cassettes . . . such as BASIC Language . . . a ham package from The Digital Group . . . and a few things like that. Much more is available in printed form . . . *What To Do After You Hit Return* is a book full of game programs and sample runs of the games to give you the idea. Digital Equipment Corporation has a book (which they don't want us to sell) of 101 games in BASIC. Much of this is pretty much the same as the *What To Do* book which is published by People's Computer Company and is available for $6.95 from *73* . . . (ahem . . . commercial).

The people who are selling programs on cassettes get bent out of shape if you run off a copy for a friend. They spent a lot of time (which is money) writing the program and getting it perfected (debugged) and they feel that you are stealing from them if you don't pay the freight. A letter was just sent around to this effect by the chaps who developed the BASIC program for MITS.

Software such as the MITS BASIC program is usually copyrighted, but this is difficult to protect. It may be that programmers will develop some smarts in this line . . . perhaps taking a hint from map publishers. Every map you see that is copyrighted has some extra squiggles in the borders of a state or a country, or something which is not there in real life. Then, when you innocently take a gas company map and trace out the part you want and use it as an illustration for your article or in a book, wham! One large map maker pulls in over a million dollars a year in copyright infringement cases, I am told. The fact is that it is awfully difficult to explain why your map has that squiggle over there or perhaps an extra town that just doesn't exist (a Cleartype Map favorite) . . . particularly when the case is scheduled to come up in court soon.

Would it be that difficult to customize each copy of BASIC with a little identifying (but non-printing or operating) character? It would be hard to find and would make each copy of the program individually identifiable . . . and might put some teeth into contracts. Just an idea . . . I'm sure ways will be found to solve the software sales problems . . . of course it might happen that programs will eventually just be part of the hardware cost and be available essentially free.

Yes, with this software you will be able to get your computer to do a lot of nice things . . . like play games. But you will still have to learn how to do your own programming for anything off the beaten track. To put it into a percentage . . . argue with me if you like . . . figure 25% hardware . . . 25% language programming and 50% your own programming efforts. Once you get into computers you are going to have your hands full . . . and you will have a ball. You may even decide to go back to fresh orange juice.

Build This Exciting New TVT

by
Louis I. Hutton K7YZZ
12235 SE 62nd Street
Bellevue WA 98006

About a year ago I decided to reactivate the RTTY mode of my station, and this article is a description of two of its facets: the keyboard and the television terminal. The keyboard development went reasonably well; then came the television terminal for RTTY, which I call the "RTTY TVT" for short. I immediately found that almost no data has been published on such a unit, with the closest thing being the computer TVT video readout devices. This problem was discussed at length both on the air and by mail with Dr. Robert Suding WØLMD, who suggested that I see what could be built using a low cost TVT circuit board kit designed by The Digital Group[1].

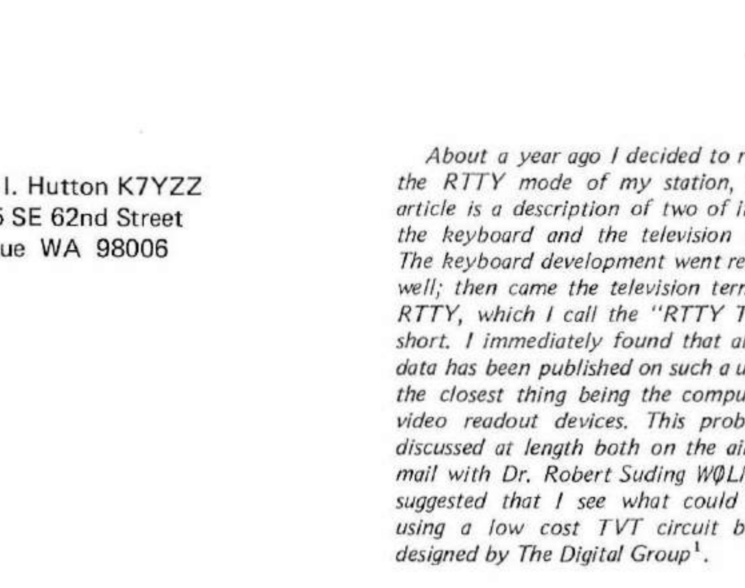


Fig. 1. Block diagram of RTTY keyboard

PART ONE KEYBOARD

Several years ago I acquired a Model 15 Teletype machine, built up a tube type terminal unit, and enjoyed many years of operation with the green keys. This equipment was retired recently and a new project was begun to update the RTTY equipment. A 100 speed Model 28 KSR was acquired and the experiments begun. This article presents an effort undertaken to build a "solid state" version of the Model 28.

A literature search was conducted to find out what has been written with regard to the design and construction of a solid state keyboard for RTTY. A RTTY keyboard using the old RTL ICs was found in an article by Krupp several years ago, and in another by Horowitz about a CW keyboard modified for RTTY operation. Another article by Bell and Schmidt was located which presented a pre-coded message on RTTY. These articles gave me some ideas on how I wanted to design my solid state keyboard for RTTY.

The unit described in this article will permit the operator to generate the RTTY Baudot code at 60, 67, 75 or 100 wpm. It features LED indication of End Of Line, Figures or Letters mode of operation. As a result all fancy frills and extras are not provided, just a keyboard, using the 7000 series TTL IC chips. This is one of the advantages of building it yourself – no unwanted expensive extras.

Circuit Description

A block diagram of the unit is shown in Fig. 1. The clock speed is manually switch selected from 60 to 100 wpm speeds. The

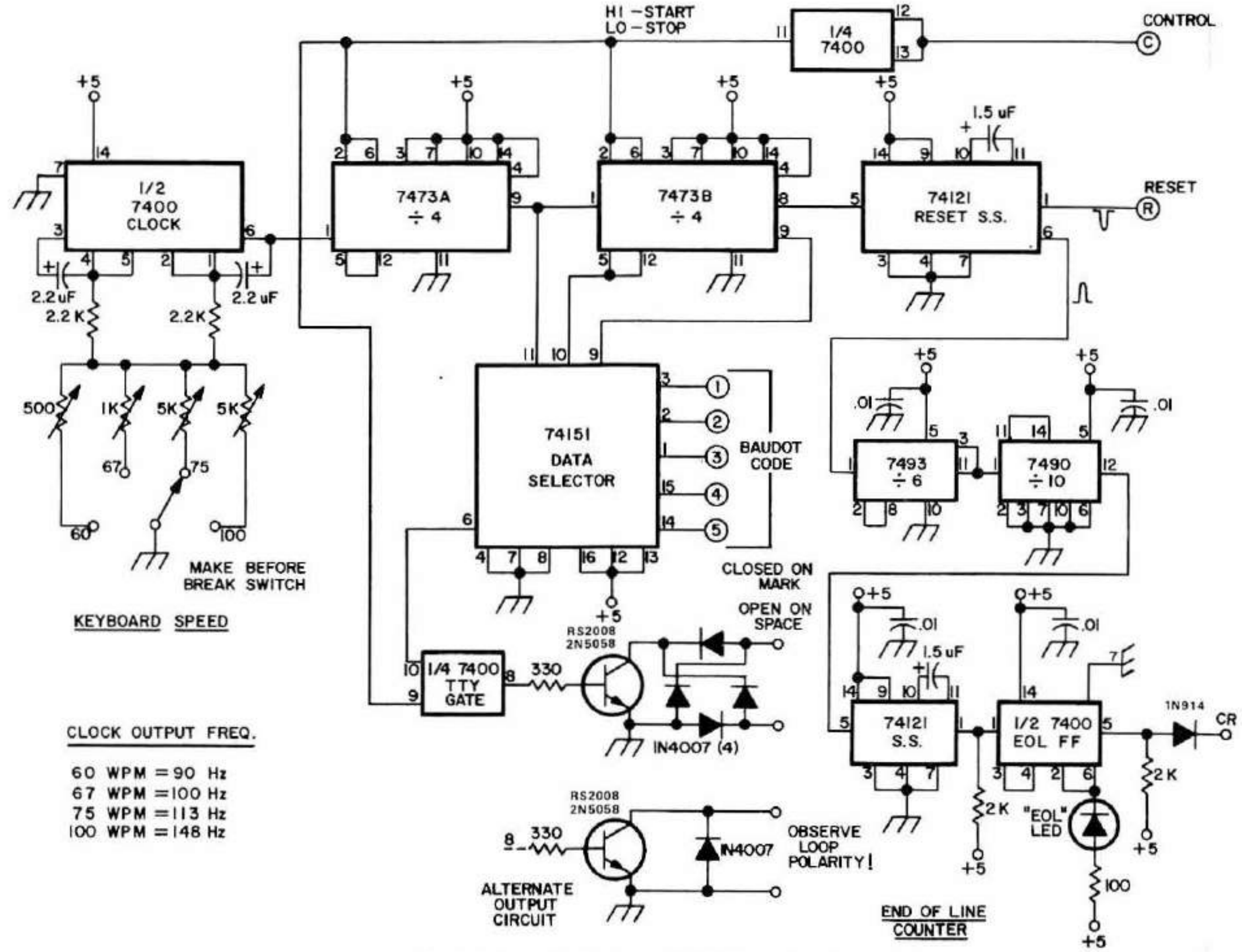
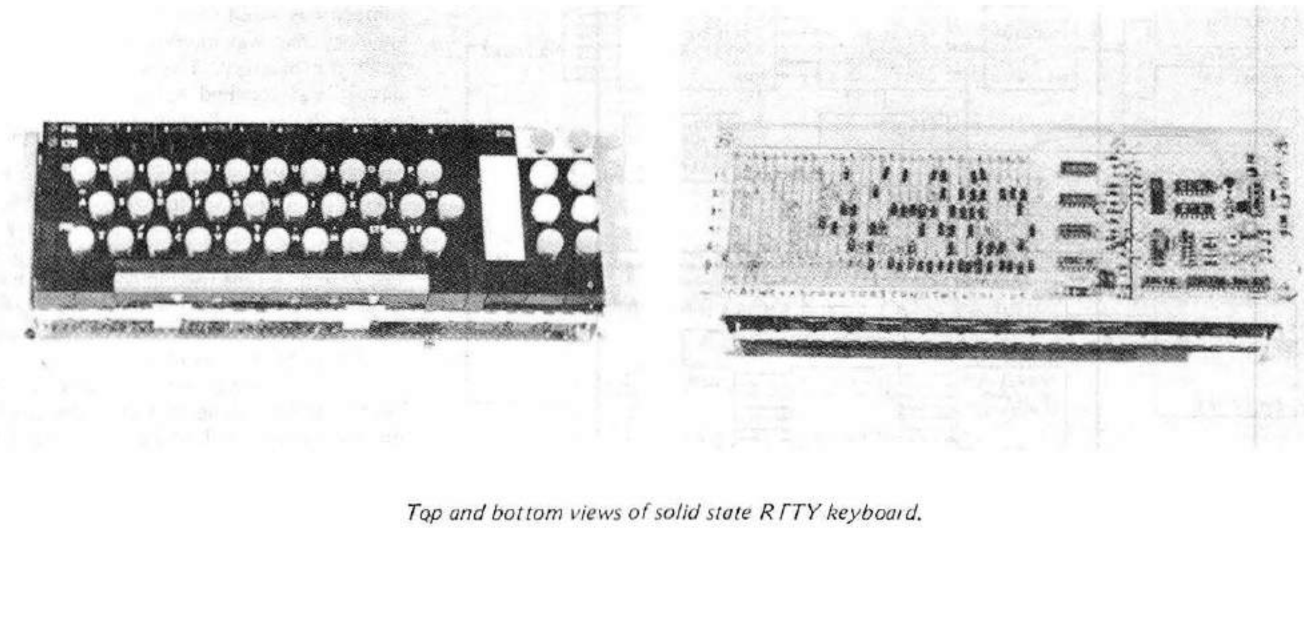


Fig. 2. Schematic diagram of RTTY keyboard.

output of the clock (90.0 Hz at 60 speed) is fed to a frequency divider chain. The three frequencies taken from the divider at 60 speed are 22.5 Hz, 11.25 Hz and 5.6 Hz. These three are fed to the Data Selector which acts like an electronic version of a TD (Terminal Distributor). When a key is depressed the Baudot code for that key is set up in the Memory flip flop via the diode matrix. The output HIs and LOs from the Memory are fed to the Data Selector. A start pulse from the keyboard is also sent to the Control FF which unlocks the frequency divider chain and the RTTY Gate. For 176 ms (at 60 speed) the frequency divider chain transmits the selected frequencies to the Data Selector which then reads out the RTTY code set up in the Memory. This code consists of a series of 22 ms pulses: one start pulse, five Baudot code pulses and two stop pulses. At the end of the pulse train the Reset SS fires and the Control FF is reset. This action clears the Memory FFs and stops the divider chain from counting any further clock pulses. The RTTY pulses are fed from the RTTY Gate to the RTTY Drive output stage which is in turn connected to a diode



Top and bottom views of solid state RTTY keyboard.

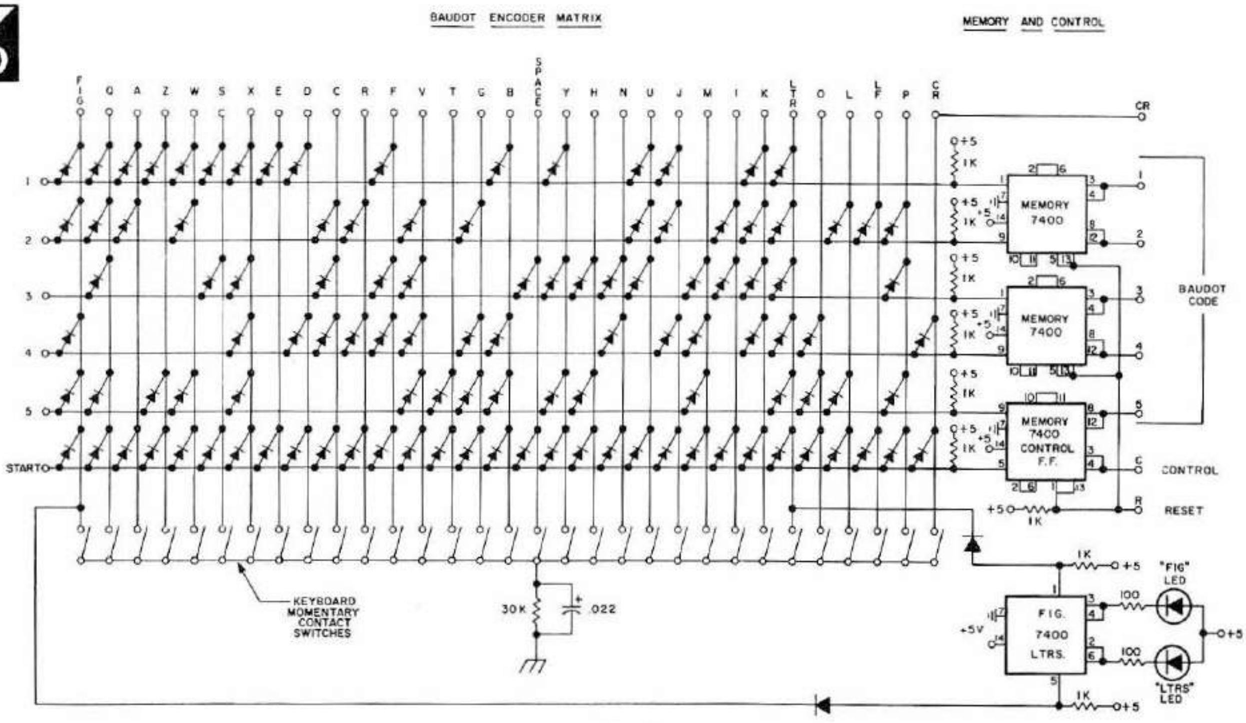


Fig. 3.

bridge permitting connection to a loop supply of any polarity. The End Of Line counter is used to indicate when 60 keyboard actions have been taken after a Carriage Return function has been keyed. This is used when the keyboard is used for driving a tape puncher. When the FIG key is depressed an FF is set and the FIG LED is lit. When the LTR key is depressed the FF is reset and the LTR LED is lit. Depressing two keys simultaneously will cause an error in the output. No mechanical or electronic means are provided in the low cost unit to prevent this event.

Construction

A surplus computer keyboard was purchased from a surplus house that advertised in *73 Magazine*. This unit uses magnetic reed switches and has plenty of extra keys for other functions that are planned for in the future. The key tops were removed and the computer markings replaced with decals of the standard RTTY upper and lower case markings. Since the keyboard already had a very nice set of numbered keys I put them on the top row, which made it not quite like a Model 15 or 28 but a bit more like a typewriter. Again, with home brewing you can make it to suit yourself.

Two metal rails were formed and attached to the bottom of the keyboard unit on metal spacers. A long piece of perforated fiberglass board 4" by 8½" was fitted in between those two rails as seen in the photographs. The diode matrix, Memory, Control FF and FIG/LTR FF were mounted on that circuit board. This configuration permitted me to conduct circuit design and debugging on a second small board containing the remainder of the circuits. The second board is 4" by 3¼". If one wishes, a board 11¼" long by 4" wide could be used for all the circuits, now that the unit has been designed. The ICs are fixed, mounted and hardwired into the circuit. Since this is not a production item no effort was made to design a PC board.

Testing

The long circuit board was wired first, and a manual jumper was wired from the keyboard key grounding capacitor and resistor to test out the diode matrix and memory before wiring keys in the circuit. A jumper was wired from the reset line on the memory that was momentarily grounded to reset the memory. The wire from the key circuit was touched to the selected diode matrix letter and then the five Baudot output lines were checked for proper high and low states. Each diode in the matrix represents a mark or high state. The absence of a diode represents a space or low state.

The diodes used in the matrix were of the fast switching type used in computers. I used some obtained from Radio Shack which carry their part number 276-114. They come in a bag of 50 each and must be tested.

When checking out the small circuit board the first thing to test is the clock. I monitored the pin 1 of 7473A to see if the clock was oscillating in each position of the speed selector switch. That switch must be of the make-before-break type for proper

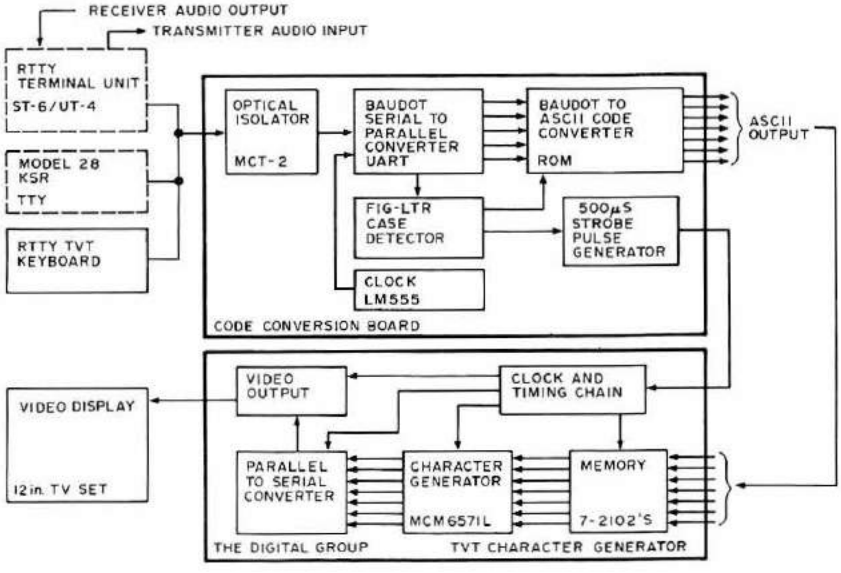
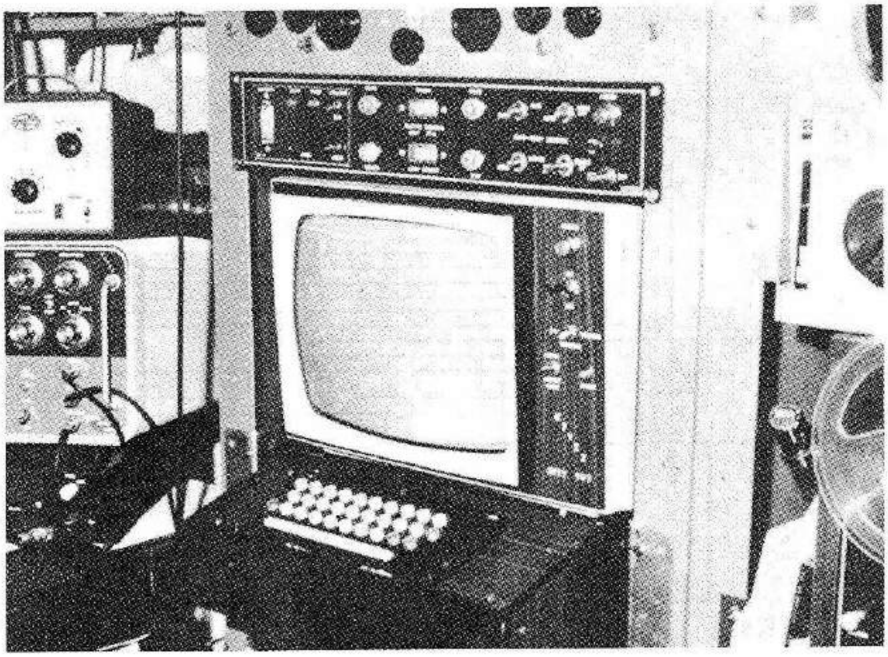


Fig. 4. Block diagram of RTTY TVT.



The RTTY TVT.

operation of the clock. Each adjustment pot was set to give the proper speed according to the chart on the schematic. At 60 speed, 90 Hz should be appearing at pin 1 of 7493A, and with the control line high at pins 2 and 6 of each of the 7493s you should get 45 Hz at pins 5 and 12 of the 7493A. Pin 9 should be 22 Hz. Pins 5 and 12 of the 7493B should be 11 Hz, and pin 8 of the 7493B should read 5.5 Hz. A negative going pulse of around a half a millisecond should be seen reoccurring each 176 ms on pin 1 of the 74121 Reset SS. With a temporary ground on pins 2 and 15 of the 74151, the output as monitored on pin 6 of the 74151 should read low, high, low, high, low, high high, repeat, etc., in a serial bit stream. If a temporary ground is placed on pins 2 and 6 of the two 7473s the bit stream should stop. If all works OK at this point, then remove all the temporary wiring and wire the board in the circuit with the encoder board.

As a final test I wired the output in series with the loop supply and printer magnets of my Model 28 KSR and was able to type out a copy on a local loop using the keyboard and the Model 28 printer at 100 speed. I ran into some noise problems and had to add some power bus bypass condensers on several of the ICs, and these are so noted on the diagram.

This unit was a very interesting project and it was a very good feeling when I finally got it assembled and running with my printer. The circuit development took several months of breadboarding to get all the various parts of the unit tested before tying them all together in the finished product. I wish to thank Mr. John Small and

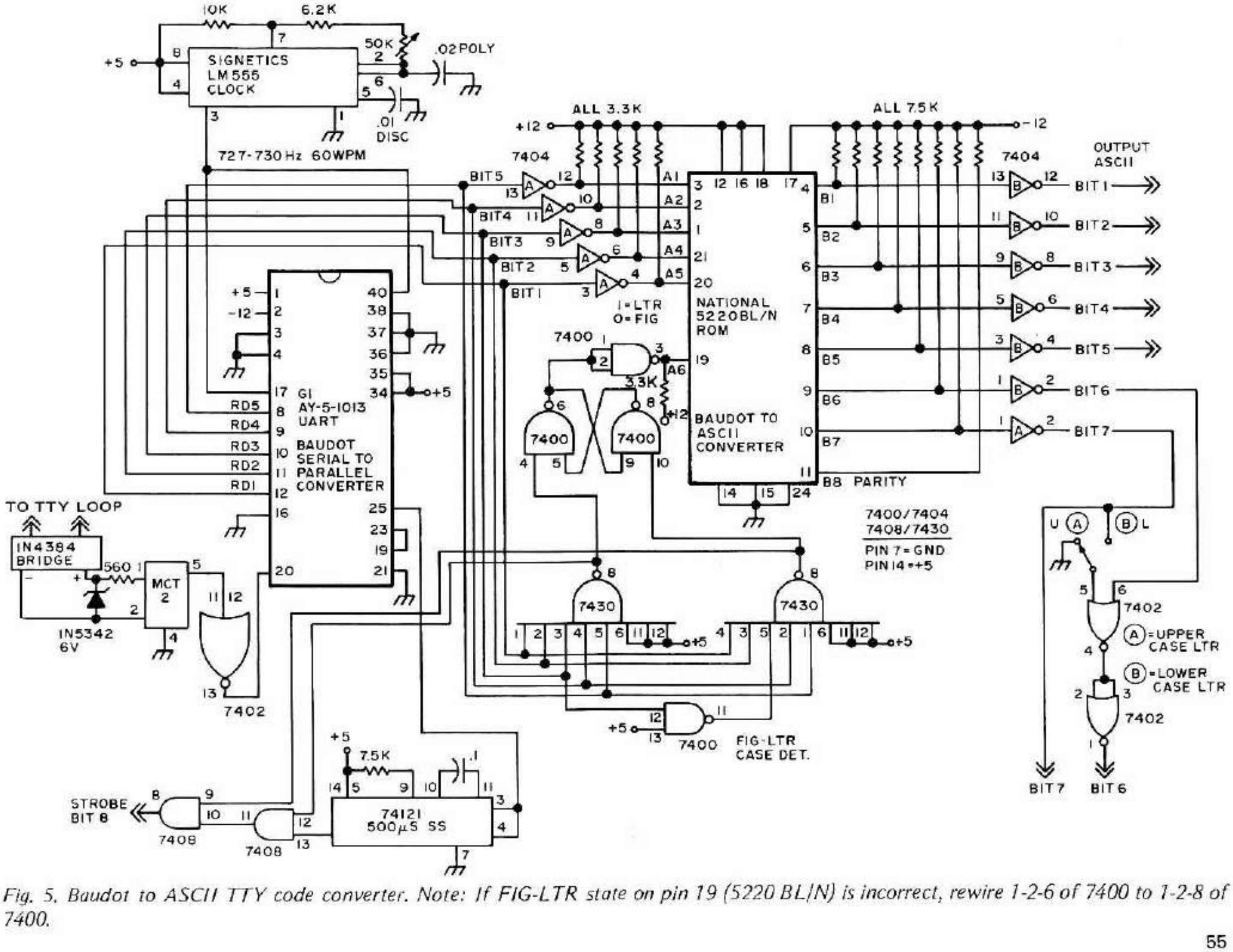


Fig. 5. Baudot to ASCII TTY code converter. Note: If FIG-LTR state on pin 19 (5220 BL/N) is incorrect, rewire 1-2-6 of 7400 to 1-2-8 of 7400.

Mr. T. Jackson who were most helpful in keeping my thinking straight as I developed this unit.

PART TWO TELEVISION TERMINAL

There are two approaches to getting that RTTY signal on a TV set. One way is to make the keyboard output the ASCII code so that it could talk to the TVT without any conversion. This method also requires both send and receive code converters to get the Baudot code converted over to ASCII to be compatible with the display. Also, when you send from your ASCII keyboard it must be converted to Baudot so the other chap's machine can print the message. The second approach is to use a keyboard that only sends in Baudot and make only one conversion from Baudot to ASCII so that the TVT could read the incoming signal from either the keyboard or the terminal unit. This is the approach that I used, and the results of my efforts are described here.

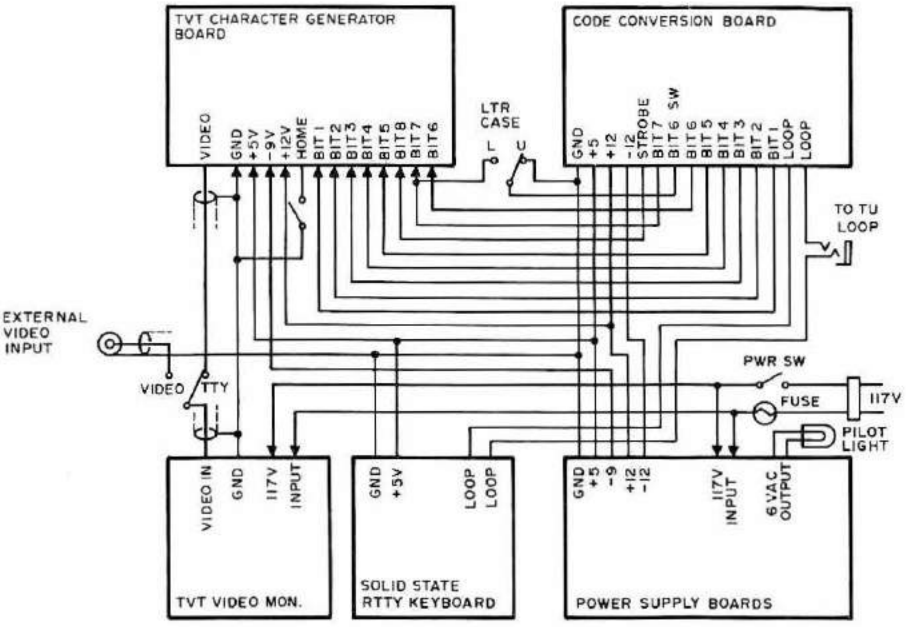


Fig. 6. Intercabling diagram.

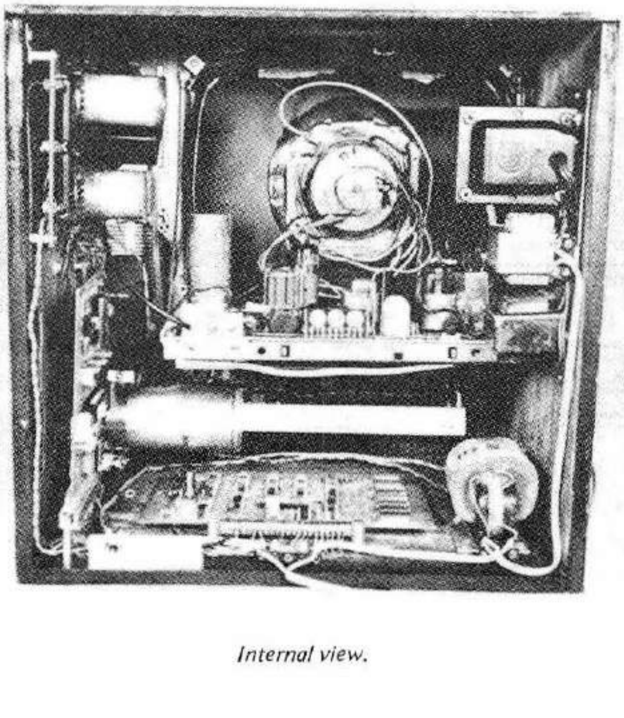
This unit includes a surplus keyboard rebuilt to include a diode matrix for encoding and generating the Baudot code at 60 wpm. The output of this keyboard is used to key the ST-6 loop and the TVT through a code converter board. This circuit board converts the Baudot encoded bit stream coming from the ST-6 or the keyboard from serial to parallel Baudot data. The parallel Baudot data is then converted into parallel ASCII data and a strobe pulse is added to trigger the TVT display circuit. The parallel ASCII is fed to the character generator circuit board and onto the TV screen. The display has eight lines of thirty-two characters per line. When the last character is printed at the bottom right-hand side of the screen the printout returns to the upper left and erases the existing character and prints the new one as it progresses across the screen line by line.

A HOME push-button switch is provided that will return the writing beam to the upper left-hand corner of the screen. An upper and lower case letter selection switch is also provided as that feature is available in the TVT circuit board by The Digital Group. Since this board was used in a microprocessor computer terminal, it has Greek notations which cause some glitches in the Baudot scheme of things. These were most troublesome in the FIG and LTR modes. The code conversion board has circuits incorporated to tell the character generator to ignore those key functions in the Baudot system. All other frills and extras were left out of this terminal to keep the cost down. It is estimated that one could probably be duplicated for anywhere from $250 to $350 depending on how many surplus components were used.

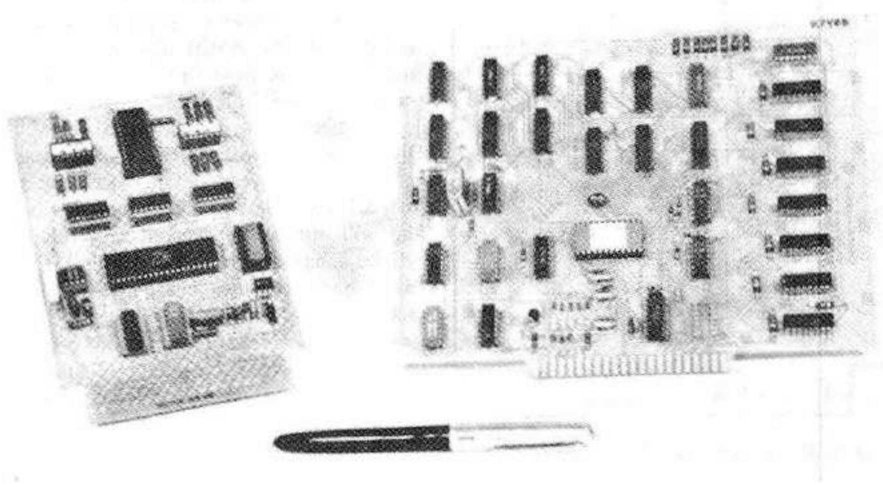
Circuit Description

A detailed description of the design and operation of the keyboard has already been presented. A block diagram of the RTTY TVT is shown in Fig. 4. The input to the code conversion board is made to the ST-6 loop circuit through an optical isolator to keep the loop high voltage from the five volt TTL circuits. An UART (GI AY51013) is used to convert the serial Baudot to a parallel Baudot output where it is fed to the input of a code converter ROM. The clock (LM555) for the UART is set for 60 wpm speed (727 to 730 Hz). In the code converter, a National 5220BL/N ROM, the Baudot code is changed to the ASCII code.[2] A figures-letters function detector is used to tell the code converter to include either bit 6 or bit 7 in the ASCII code output. The case detector is also used in conjunction with the strobe generator to eliminate the Greek letter that would normally appear on the TV screen when the FIG or LTR key is touched. A strobe of at least 500 microseconds is required at the end of the generation of each character on the TV screen. The strobe generator is triggered by the output of the UART transmitter section.

The ASCII output, along with the strobe line, is connected to the TVT character generator board. The video output from the TVT board is connected to the video input of a modified TV set. To prevent the TVT display from returning to home position after every time a CAR RET is keyed, IC30 (7430) is removed from its socket. A 2.2k pull up resistor is connected between pins 8



Internal view.



The two PC boards.

and 14 of the IC30 socket. A wire is run from pin 8 of IC30 socket to pin 14 of the PC board connector. This is the reset "HOME" terminal. When this terminal is grounded momentarily via a push-button switch it drives the write beam of the CRT to the beginning of the first line on the display. This is the only modification required to The Digital Group TVT circuit board to get it working as a RTTY TVT circuit board. The video readout of the RTTY TVT unit is a salvaged 1969 Sears 12" portable TV set. This set came from a junk yard, and to put it into use as a video monitor all the circuits were removed except the sweep, video and high voltage. It makes a very acceptable video monitor for this application.

Construction

The cabinet for the RTTY TVT is constructed from plywood and is 15" high by 15" wide by 15" deep with about 4" sticking out in front to house the keyboard. The outside of the cabinet was sanded and given two coats of shellac. The edges of the cabinet were painted a dull black. The remainder of the exterior of the cabinet was covered with simulated wood grain sticky-black plastic covering obtained from a hardware store. This makes a really nice looking cabinet. The panel holding the keyboard was also covered with this same simulated grain material. A metal panel was fitted over the holes left by the removal of the TV tuners and controls, and was covered with the simulated wood grain material.

The power supplies were constructed from surplus parts and are mounted by metal standoffs to the inside of the cabinet. The circuit board containing the code conversion circuit is 4½" wide by 6" long. The TVT readout board is 8" wide by 6½" high. A diagram of the code conversion board is shown in Fig. 5. The overall intercabling diagram for the RTTY TVT is shown in Fig. 6.

The conversion of the 12" TV set for use as a video monitor required considerable time as the set was in pretty bad condition. The filaments of the four tubes used were rewired so that they would work from lower voltage circuits. I did not want to run the thing from the power line direct like it was originally wired. I used a small isolation transformer to supply the 120 volts for the power supply input. I found that both the yoke and flyback transformers were defective and had to be replaced using parts from other discarded sets. Fig. 7 is a schematic of how I accessed the input to the video amplifier of the rebuilt set. The original set used a tube to rectify the high voltage to the picture tube and I replaced that with a solid state high voltage rectifier removed from another junk set. This helped to further lower the current drain on the set. The front part of the old set's cabinet formed the front of the TVT unit. It was attached to the plywood cabinet by hand-made metal brackets. The panel to the right of the picture tube contains all the controls for operating the RTTY TVT. These are the pilot light, power switch, video monitor or RTTY monitor selector switch, upper case-lower case letters selector switch, and the HOME switch. The CRT brightness and contrast controls were mounted on a small metal plate fastened to the rear of the TV chassis. They are set once and left that way.

Test and Checkout

Since I was developing many of the circuits to put this thing together, I ended up by running all sorts of tests during that time. Many of them are not needed for someone essentially duplicating this circuit. When the code conversion board is finished, the input to the board should be connected to the output of the keyboard and the ST-6 or terminal loop supply. This way you can send Baudot encoded characters to the board one at a time to see if they are being encoded properly at the output of the board in the ASCII code. The clock must be set for an output frequency of from 727 to 730 Hz as measured at pin 3 of the LM555 (60 wpm). Each time a key is depressed on the keyboard a pulse of approximately 500 microseconds should appear at pin 8 of the 7408 gate. There should be no pulse appearing at that point when the FIG or LTR keys are keyed. A "high" or 1 should appear on pin 3 of the 7400 when the LTR

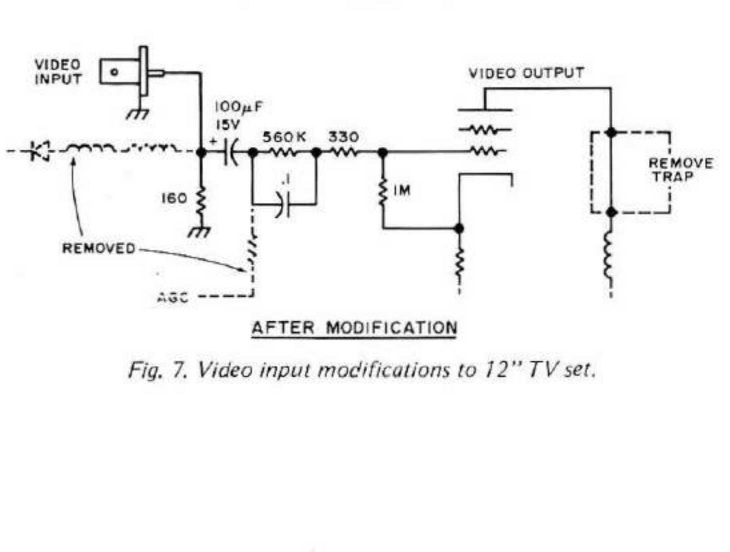


Fig. 7. Video input modifications to 12" TV set.

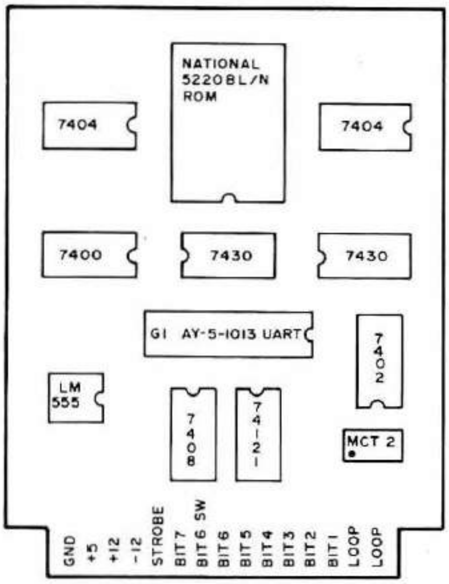


Fig. 8. Major parts layout, Baudot to ASCII code conversion board.

key is depressed, and a "low" or 0 should appear on pin 3 of the 7400 when the FIG key is depressed. Pin 5 of the 7402 should be temporarily grounded so that the output of the code converter may be checked.

Key in the letter A on the keyboard. Now look at the output of the board and bit 1 and bit 7 should be high and all other bits low. Now try the letter R on the keyboard. Only bits 2, 5 and 7 should be high on the ASCII output. Now try the letter Y and the ASCII readout should have 1, 4, 5 and 7 high. Now key the FIG key and then the number 2. The ASCII output should have only 2, 5 and 6 high. On FIG operation bit 6 of the code is used and on LTR bit 7 is used. If at this point you are getting this kind of response on the ASCII output you are in fine shape. The next thing is the character generator circuit board.

The circuit board data furnished by The Digital Group with their TVT character generator gives very detailed step by step instructions on how to check out their board and will not be repeated here in this brief article. I had an old 1950 Admiral TV set that I had previously modified for use as a video monitor with a camera for fast scan to slow scan conversion construction project. I used this set to test out the TVT board and to check out the system from keyboard through the code converter through the TVT board and onto the TV set. This way I was sure all bugs were worked out before I mounted the parts in the cabinet. The TVT circuit board performed as designed with only a slightly defective memory chip giving me any problem. This bad chip caused the wrong character to appear in certain lines on the screen. When replaced by the vendor it worked perfectly.

When connecting the input of the RTTY TVT into my ST-6 I had a polarity problem and had to reverse the input leads to the RTTY TVT to get it to print properly. This was the only interfacing problem encountered. I have my Model 28 KSR wired in series with the loop driving the RTTY TVT and if I want a hard copy I just turn on the power switch on the machine motor and the copy is printed.

I would be interested in hearing from those who decide to build one of these RTTY TVT units and will try to answer any questions they may have about the machine. All I ask is that they include an SASE. 73s and good printing on your CRT. ■

References

1. 256 Character Display Kit, The Digital Group, P.O. Box 6528, Denver CO 80206.
2. *TTL Cookbook,* Don Lancaster, 73 Radio Bookshop, Peterborough NH 03458.
3. "RX for the RTTY Bug," Louis Hutton, *73 Magazine,* February, 1962.
4. "Attache' Case RTTY," D. Krupp, *QST,* February, 1968.
5. "Perfect Teletype at Your Fingertips," P. Horowitz, *QST,* October, 1968.
6. "A TTL Message Generator for RTTY and CW," J. Bell and F. Schmidt, *QST,* November, 1973.
7. "TV Typewriter," Don Lancaster, *Radio Electronics,* September, 1973.
8. "TV Typewriter II," Ed Colle, *Radio Electronics,* February, 1975.
9. CT-1024 Terminal System, Southwest Technical Products, 219 West Rhapsody, San Antonio TX 78216.
10. "The Scopewriter," Gary Steinbaugh, *Popular Electronics,* August, 1974.
11. RVD 1002 Visual Display System, Hal Communications Corp., P.O. Box 365, Urbana IL 61801.
12. "Television Interface," Don Lancaster, *BYTE Magazine,* October, 1975.

EDITORIAL

BILLIONS . . . BILLIONS!

It isn't often that we can see quite so clearly a new market that is inevitably going to burgeon. Usually, when you can see something like this coming, you can use the information to make some money . . . like investing in Haloid just before it became Xerox. The coming explosion in computers . . . which will dwarf the already well-known computer growth . . . does not give us any clear insights into turning a fast buck, but it certainly can be used to advantage.

I've already suggested that amateurs, by virtue of their head start in electronics, will have a decided advantage if they want to get into the sales or service of small computer systems. This is still a bit in the future, despite the opening within the last few months of a couple dozen computer stores around the country. Right now most computer stores are serving hobbyists rather than small businesses. This is primarily because to date there are really no practical small business systems available, only hobby systems. But business systems will be along . . . and shortly.

How much of a business will there be? I would think we could expect at least a $50 billion a year volume . . . and that probably is conservative. When computers get down to the $250 to $500 range, and they will soon, you will be seeing them everywhere . . . on every desk at school . . . every desk in every office . . . in every small business or store . . . in every home. They will be used for writing letters (most of which will eventually go via cable or satellite), keeping records, ordering from the store, billing, keeping bank records, teaching, games, art, music composition, newspaper replacement, etc.

If you are not really interested in getting into manufacturing or sales, you may want to look for investment in firms which are getting into these fields. While it is still too soon to get even a faint idea of how the pioneers in the field are going to do, you could do worse than keep a good eye peeled for up and coming firms.

uP AND AMATEUR RADIO

For some reason I've had a lot less static about the coverage of computers in *73* than I expected . . . less than when I got started on FM and repeaters some years ago. While I recognize that a great many readers understand the impact that computers are going to have on amateur radio, still I also realize that despite our efforts to keep the readership of *73* confined to active and progressive amateurs, some spark-forever types are bound to have infiltrated.

Just as many amateurs reacted violently to transistors, refusing for years to accept that they were here to stay (see the *QST* George Grammar editorials in the mid-60s). I expect this same reaction to the revolution microprocessors are bringing us. The idea of a small PC board with a few chips being able to replace very complex electronic systems is going to take some getting used to.

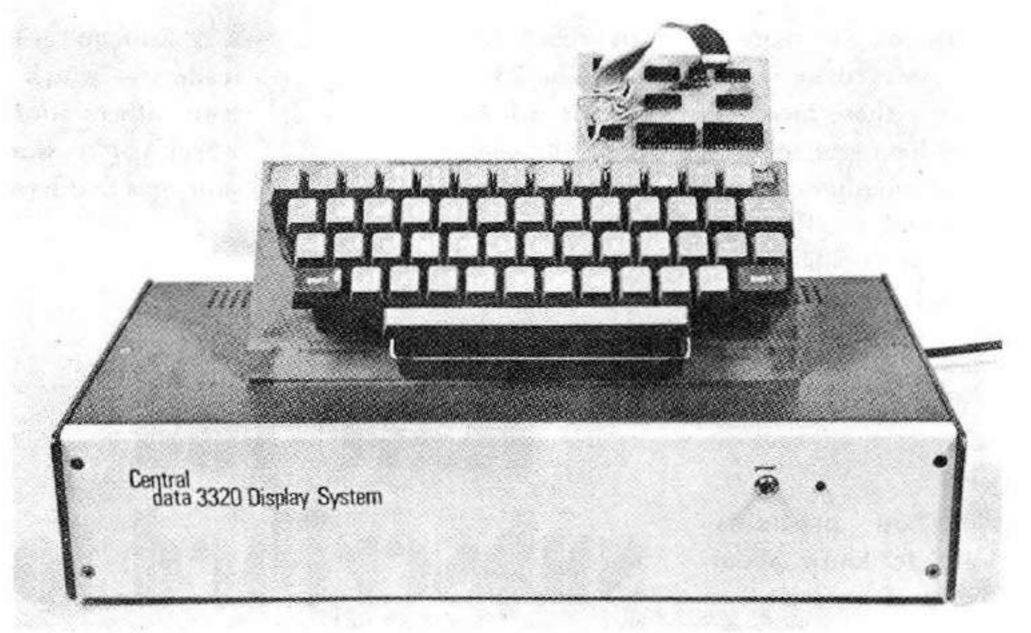
How can an old timer (of any age) get used to a tiny computer system which can be used to change typing into Morse code, into RTTY, for editing text to be typeset, for playing games, for teaching almost anything, for keeping a mailing list, for running a repeater, or for just about anything the mind can imagine? It is almost too big a step to really comprehend. Just by changing the programs in their computers, two amateurs can use their systems to communicate by Morse, RTTY with Baudot code, RTTY with ASCII code, or any other agreeable system. With a slightly different program the system will see what code is being received and cope with it automatically. Pity the old timer.

Since I'm as new at all this as you are, I'll be trying to bring you articles which will help us to learn . . . and as I begin to understand more of what is going on I'll try to keep you in touch. These new, relatively inexpensive, computer systems are a quantum jump for us, so we've got our work . . . and fun . . . cut out for us.

CONVENTIONS

The computer hobby field is getting big enough so they are already starting to hold conventions. An unusually high percentage of the computer hobbyists seem to also be radio amateurs, so you might find an event like this fun.

The first such convention, run by MITS at their plant in Albuquerque, is scheduled for the end of March . . . as this issue goes to press. I'm planning to be there and report on this convention. The next one is May 2nd in Trenton, New Jersey and is being organized by Al Katz K2UYH, a well known UHF pioneer and Oscar addict. I'll try to be there and report on that meeting, too. More than a thousand hobbyists are expected to attend. There will be manufacturers' exhibits, prizes, demonstrations of hobby systems, plenty of technical talks and a flea market. Write Trenton Computer Festival, Trenton State College, Trenton NJ 08625 for details.



Jeff Roloff
Central Data Company
PO Box 2484, Station A
Champaign IL 61820

Photos by Ed Crabtree

A RTTY/Computer Display Unit

--Baudot, ASCII, TTL, RS232, etc., etc.

In order to interface man with computer, some sort of input/output device is needed. An extremely versatile output device is a visual display unit, which displays data on any modified television screen.

The display unit described in this article is very efficient, both in price and in the density of the data on the screen. It displays 25 lines of 40 characters each on the screen – almost double the density of most other display units now on the market. And the parts cost is less than $100.00, buying from companies advertising in the back of this and other electronics magazines, including the price of the double-sided, plated-through board. All display unit parts, including the $20 power supply, are contained on one 7.2" x 11.4" board. The option boards are both about 4" x 4".

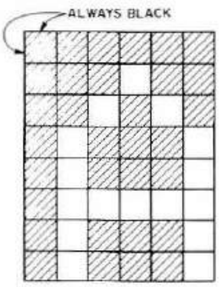


Fig. 1.

The two option boards that are available are used to input serial ASCII and serial Baudot. The first accepts the serial ASCII, converts it to parallel, and enters it into the display unit's memory. It also accepts parallel data from a keyboard, etc., and serializes it. The serial interfaces can be either TTL, EIA RS-232C, or 20 mil loop compatible, all selected by two jumpers. The other option accepts serial Baudot code, converts it to parallel, and changes the code to ASCII. This ASCII data is then sent to the display unit. The inputs to the Baudot board can be either TTL, EIA RS232C, or 60 mil loop.

A useful feature of the display unit is called next line blanking. This causes the screen to be erased, line by line, as the cursor moves down a page full of data. If this feature was not in the display unit, after the screen had been filled with data once, you would simply be writing over old data, character by character. To say the least, this can be very confusing! Full cursor control is available using the "forward/backward" input. The cursor can be moved straight up or down, left or right, or to the left of the screen, and then up or down one line. A self-test line is available so that signals can be displayed on the screen; therefore, in most cases, an oscilloscope is not needed. All logic is done with standard TTL gates and dividers. There are no capacitively coupled lines or other bears to troubleshoot, so a person with a modest experience with logic will probably be able to easily fix his board, should it not work.

Basic Information

The 3320 display unit has a format of 25 lines consisting of 40 characters each. The characters are arranged in a standard 5 x 7 matrix, each space in the matrix being either a white or a black dot on the screen of the television. There is a one dot space both horizontally and vertically which is always

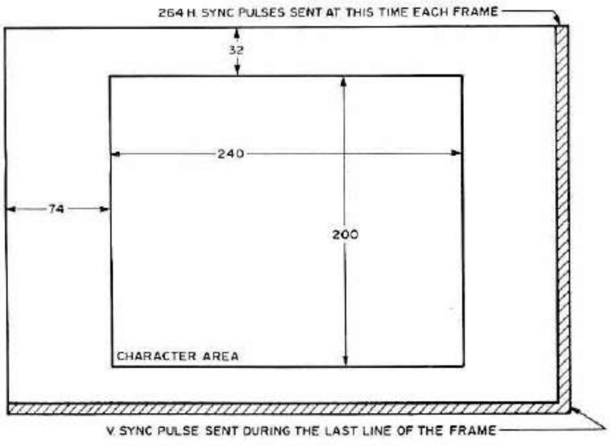

264 H SYNC PULSES SENT AT THIS TIME EACH FRAME
32
240
74
200
CHARACTER AREA
V SYNC PULSE SENT DURING THE LAST LINE OF THE FRAME

Fig. 2.

black to make sure that there is space between characters. Therefore, each character takes up a space of 6 x 8 dots on the screen, as shown in Fig. 1 for the letter "A". Each dot represents one cycle of the master clock.

The cursor, or the marker showing the next character position, is a white line five dots wide on the top scan line of that character, which is otherwise always black. The cursor flashes at a rate of two times per second to aid in finding it in a crowded screen.

Since characters take up 6 x 8 dots, and there are 40 characters on each of the 25 lines, there is a display area of 240 x 200 dots. There are provisions in the timing chain, however, for 384 x 264 dots. The reason for this is that the television's beam scans past the edge of the screen, where you can't see it. When it is well past the visible area on the screen, a horizontal or vertical sync pulse is sent which causes the beam to go back to the left edge or to the top of the screen, respectively. The display is centered within the 384 x 264 dot matrix, and fills up the visible screen of most television sets. This arrangement is shown in Fig. 2.

To interface the unit with the outside world, a parallel input and video output are provided. The input consists of seven data lines coded in ASCII, along with a strobe line. The strobe line is used to tell the display unit when to enter the data. When the strobe line makes its transition (either positive or negative, jumper selectable), the data is entered into memory. This strobe line is de-bounced on the board, and a $\overline{\text{strobe received}}$ pulse is available at the output connector. This is used by the input options and is required by some types of keyboards. The video output is a 2.25 V P-P signal with the specifications shown in Fig. 3.

Timing

The schematic diagram for the timing chain is shown in Fig. 16, while the timing chain is shown in Fig. 15.

All timing is derived from a crystal-controlled oscillator whose frequency is 6.082560 MHz. The output of this oscillator is called the A clock, and is fed into various dividers until it finally reaches the T clock, being 60 Hz.

The B clock is exactly half of the master clock, since it goes through the first flip flop in the divider chain. The C, D, and E clocks are all of the same frequency, but are different in phase. Since characters are six dots wide, there are six A clock pulses, three B clock pulses, and one each C, D, and E clock pulses. This is shown in Fig. 4 for the top of the letter "A".

The F clock is half of the D clock, and from the G to the J clocks, the previous clock is divided by two to get the present one.

The K clock is special, however. It is low when characters can be printed across the line and goes high when they can't (during horizontal retrace). There are enough clock pulses for 64 characters on a line, but only 40 of these are used. The extra 24 character positions cannot be displayed on most television sets, anyway, and are used for horizontal retrace. The K clock is not the output of a flip flop, but rather a latch that gets set on the 41st character and reset on the first character. In the middle of retrace (when the I clock is low), a horizontal sync pulse is created. This tells the television to send the electron beam back to the left side of the screen so that a new scan line can begin.

The K clock is divided by two, four and eight to get the L, M, and N clocks, respectively. These three clocks are fed to the character generator to tell it which scan line of the character the television is displaying. This is shown in Fig. 5 for the left side of the letter "A".

Note that the top scan line is always blank. This is to provide vertical spacing between characters, and to allow time for 40 characters of the page memory to be moved into the line memory.

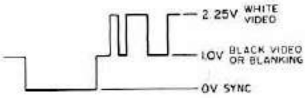


Fig. 3.

This is explained in detail in the memory section.

The last six clocks are different because they are reset at the end of the frame. There are provisions for 264 scan lines per frame (33 character lines of eight scan lines each). Since all dividers must be synchronized with each other, all affected counters are reset at the end of the frame, and the whole timing chain starts over.

As was mentioned earlier, there are provisions for 33 character lines, but only 25 of them are used. The T clock is low when characters can be vertically printed on the screen, and high when they can't (during vertical retrace). The T clock is like the K clock, in that it is a set/reset flip flop arranged as two cross-coupled NOR gates. The flip flop is set on the 26th line and reset on the first. In the middle of vertical retrace, the vertical sync pulse is created, telling the television's electron beam to go to the upper left hand corner of the screen and start a new frame. Several other timing pulses are derived from the lettered clock pulses, and they are explained in the cursor control and memory sections.

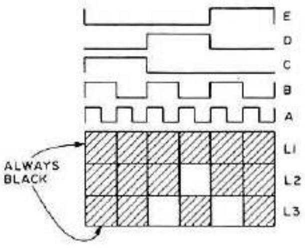


Fig. 4.

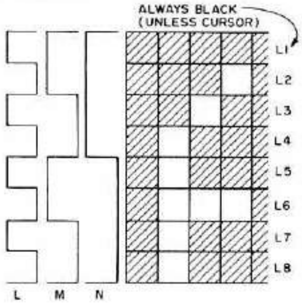


Fig. 5.

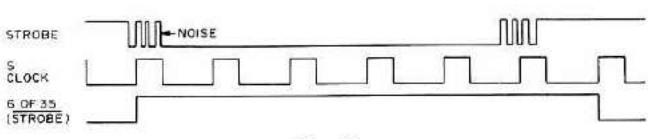


Fig. 6.

Bits b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4 ↓	b3 ↓	b2 ↓	b1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	\	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	8	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	Λ	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

ENQ	– Enquiry	ETB	– End of transmission block	SYN	– Synchronize
ACK	– Acknowledge			FS	– File separator
BEL	– Bell	CAN	– Cancel	GS	– Group separator
BS	– Back space	EM	– End of media	RS	– Record separator
HT	– Horizontal tab	SUB	– Substitute	US	– Unit separator
NUL	– Null	ESC	– Escape	LF	– Line feed
SOH	– Start of heading	SI	– Shift in	VT	– Vertical tab
STX	– Start of text	DLE	– Data link escape	FF	– Form feed
ETX	– End of text	DC1-DC4	– Device controls	CR	– Carriage return
EOT	– End of transmission	NAK	– Negative acknowledge	SO	– Shift out

Fig. 7.

Memory

The memory schematics are shown in Fig. 19.

There are two clocks associated with the page (hex-1024 bit) memory, and one more is created for the line (hex-40 bit) memory. The main memory clocks are labeled Ø1 and Ø2, Ø1 being the output clock and Ø2 being the input clock. For every character that is shifted, both clocks have to be sent to the shift registers. These two clocks have to swing from +5 to -12, which are not standard TTL levels, and must not overlap. There is a clock driver between the timing chain and the memory to buffer these clocks. Since there are 1024 positions in the shift register, but only 1000 are used to display data, 24 extra clocks must be added somewhere in the frame.

These extra 24 pulses begin at the rise of the T clock, and end after 24 pulses have been counted by circuits 21, 13, and 22. These 24 pulses are logically ORed with the 1000 pulses that occur normally. The 1000 pulses occur during the scan line that is always black on the screen (L1), and only in the character area (the 240x200 dot area where characters are displayed). There are 40 of each clock per character line, where a character line consists of eight scan lines.

The line memory is clocked by the line clock which comes 40 times per scan line. During the top scan line of a character line (L1), the page memories are being clocked out, and that data is being clocked into the line memory. The data that was in the line memory is sent to the character generator, but since it is scan line number one, the output of the character generator is all zeros. On the next seven scan lines the data just fed into the line memory is clocked into the character generator over and over again.

The character generator is actually a read only memory that is addressed by the output of the line memory and the L, M, and N clocks which tell it which scan line the unit is now displaying. The output for the first scan line (L, M, N = 0) is always zeros, so the top scan line always is black. The output for the second scan line of the letter "A" would be 00100, the one being the point on the top of the letter.

The output of the character generator is sent to a shift register, which shifts the data out at the frequency of the A clock. The serial input of this output register is grounded so that during retrace its output is low, making black dots on the screen. The character generator's output is loaded into the output register when $\overline{\text{load}}$ is low and the A clock rises. Note that the first load pulse is created one character position after the other memory clocks start. This is because there is a long propagation delay between the page memory's and the character generator's outputs. The output of the shift register is sent to a buffer transistor where it is mixed with the sync pulses and is sent to the television.

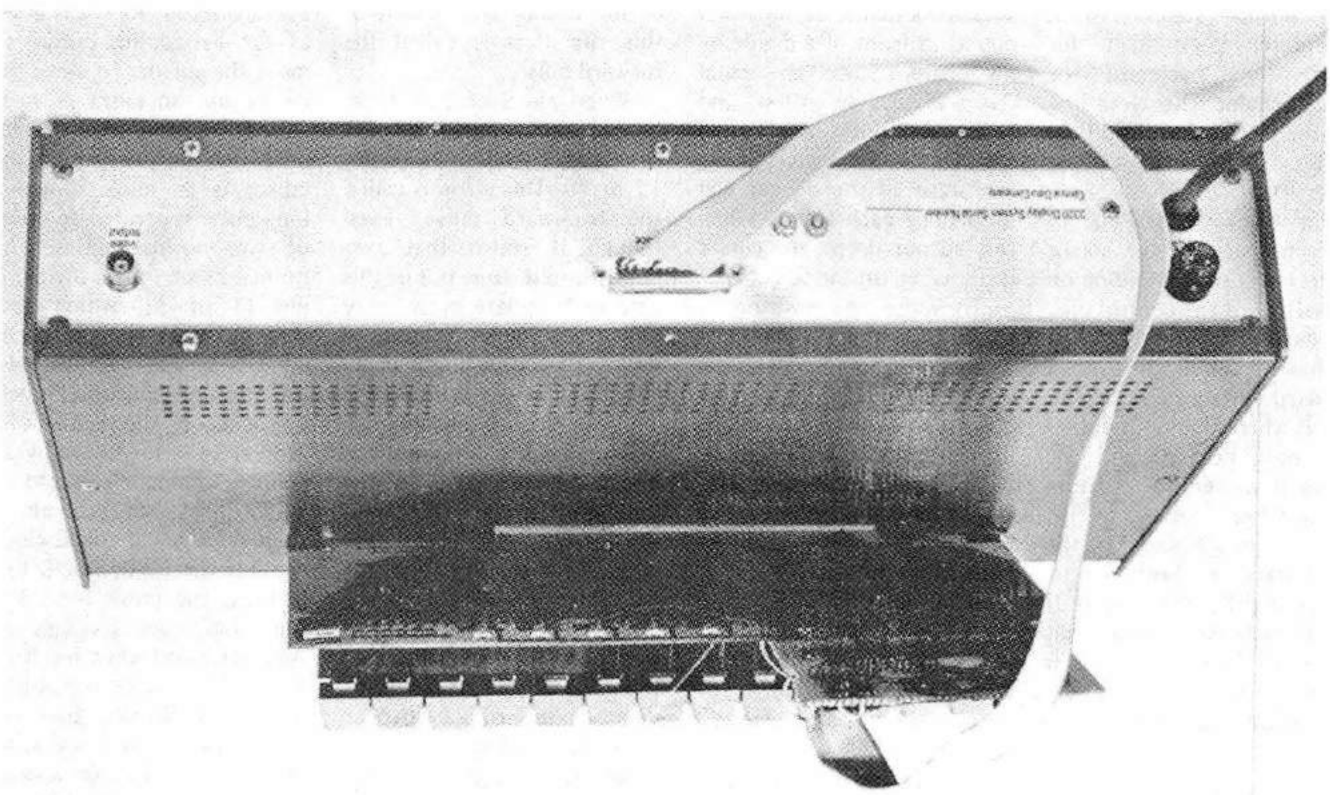
Input Conditioning

The input conditioning and cursor control schematics can be found in Fig. 17.

In order to enter the input data into the memory, it is required that a strobe pulse accompany the data. This pulse can be either negative or positive going, depending on the wiring of the strobe jumper. If it is a negative strobe, points A and C are jumpered as explained on the cursor control drawing.

The de-bouncer is the first type D flip flop in the chain. After strobe is received (pin 2 of 35 goes low), the device waits for the next rise of the S clock. When this happens, the output (pin 6) goes high. The reason that this circuit de-bounces is that the S clock has a period longer than any noise on the strobe pulse could last.

The sequence of events is shown in Fig. 6. Pin 6 of 35 is sent to the clock of the next flip flop. When it goes high, the grounded input is transferred to the output (pin 9), and that sets the NAND gate set/reset flip flop. When this flip flop's Q output (pin 8) goes high, its $\overline{Q}$ output goes low, running back through an AND gate and setting the second type D flip flop. Even though pin 9 of 35 now goes high, the set/reset flip flop stays set (pin 8 stays high). That output is sent to the input of another type D flip flop whose clock is the $\overline{T}$ clock. Therefore, when $\overline{T}$ goes high, pin 5 of 36 goes high and 6 of 36 goes low. Pin 6 of 36 is fed back to reset the set/reset flip flop, bringing pin 8 of 17 low. The next time the $\overline{T}$ clock rises, pins 5 and 6 will go back to



their normal states. Pin 5, therefore, goes high for exactly one cycle of the timing chain. The clear line sets the second and third type D flip flops to the correct states.

Pîn 5 of 36 is sent to different parts of the cursor control section depending on the setting of the control bits. If bits 6 and 7 of the input data are not both low, a standard update cycle begins. This is where the input data is put into memory and the cursor is moved one position forward or backward. If, however, bits 6 and 7 of the input data are both low, a line feed (LF) cycle begins. If an LF cycle happens with bit one low, then a line feed is initiated (the cursor moves straight up or down one line).

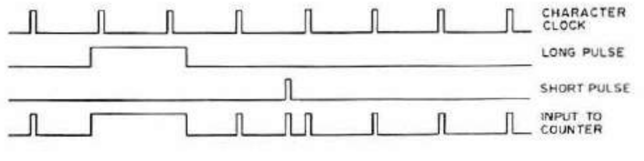


Fig. 8.

If bit one is high during an LF cycle, the cursor moves back to the beginning of the line, and then up or down one line. In this case, a set to 40 pulse is created at pin 11 of 38 since bits 6 and 7 are both low (1 of 37 is high) and bit one is high. This status is ANDed with the up or down pulse (only created during an LF cycle) to arrive at the set pulse. The any cycle line is hooked directly to pin 5 of 36, so it goes high for one cycle after strobe drops no matter how any of the data bits are set. Fig. 7 shows the ASCII codes as used in the 3320 display unit.

Cursor Control

The cursor control section can be thought of as a chain of flip flops that divide their input by 1000. The clock for this divider sends a pulse during the first scan line (L1), above where each character could be. The divider is actually a divide by 40 chain placed in front of a divide by 25 chain. Since the total chain divides by 1000, and 1000 clock pulses are created each frame, the output of the last stage of the divider will drop once each frame. When this output drops, the cursor is displayed on the screen.

To move the position of the cursor, the divider must drop at a different time in the frame. This can be accomplished since the input of the divider can receive pulses from either the character clock generator or a special circuit that either adds or takes away pulses.

If an extra pulse is fed to the divider, the output will drop one character earlier. If a pulse is taken away, the output will drop later, since it will take longer to count 1000 pulses. Instead of taking away pulses, a long pulse is created that "covers" two normal pulses, as shown in Fig. 8. So if we want to move the cursor back one position, an extra pulse is added between the normal character clock pulses. If we want to move the cursor ahead one position, we cover up two pulses by adding one long pulse. The pulse, whether short or long, is created when the update cycle drops, setting pin 9 of 22 high. Pin 9 of 22 goes low the next time that the E clock rises, and will stay low until the update cycle drops again. If the FWD/BKWD line is high, this pulse goes through an AND gate to set a set/reset flip flop, IC23. The next time that the G clock goes high, the set/reset flip flop gets reset. This flip flop, therefore, stays set for the first two character positions of the frame. The output of this flip flop is called the forward pulse.

When pin 9 of 22 is high, an A clock pulse is sent through an AND gate to pin 12 of 20. This pulse is called the backward pulse. Even though it seems that two pulses would come out of this gate each update cycle, only one does. This is because there is a propagation delay between the A clock and the T clock. Because of this small delay, the T clock (and the update cycle) drop in the middle of the first C clock of the frame. Therefore, the very first A clock pulse of the frame does not get through to pin 12 of 20, but the second one does. By the time that the third A clock pulse comes, pin 9 of 22 has been reset to 0. The one short pulse that does get through is called the backward pulse.

The character clock, forward pulse, and backward pulse are all ORed together to form the input to the divide by 40 chain. If there wasn't just an update cycle, only character clock pulses get through. If there was just an update cycle and the FWD/BKWD line is low, the backward pulse will also get to the clock of the divider. This will clock the divider one extra time, so its output will drop one position earlier. If there was just an update cycle, and the FWD/BKWD line is high, the forward pulse gets through and covers the first two character clock pulses as well as the backward pulse. There is one less clock pulse this frame, so the divider drops later.

The divide by 40 chain consists of circuits 30 and 31, and its output is pin 11 of 31. When this output drops, it clocks pin 6 of 32, setting pin 11 to a low state. On the next character clock pulse this flip flop is set (pin 11 goes high) and stays that way until the counter drops again.

The divide by 25 chain operates the same way that the divide by 40 chain does, as far as adding pulses to move the cursor. To move the cursor up, an extra pulse is added, making the divider drop one line earlier. If the cursor is to move down, a long pulse is created to cover up two normal pulses. The normal clock to this divider is pin 11 of 32, which goes through IC28 to get to the divider (circuits 33 and 34). The dropping of the LF cycle line sets off the extra pulse generators in the divide by 25 counter chain. It sets pin 12 of 24 high, but that pin is soon reset when the E clock rises. If the FWD/BKWD line is high, the pulse from this flip flop goes through an AND gate and sets a flip flop, circuit 27. The $\overline{Q}$ output (pin 4) of this flip flop goes low immediately and rises again when the P clock goes high. The $\overline{Q}$ output of this flip flop is therefore low for two character lines (16 scan lines), and is called the down pulse. Pin 12 of 24 also is ANDed with the A clock to create a short pulse (the up pulse). The reason two pulses are not created is the same as was explained for the divide by 40 chain.

The up and down pulses, along with the normal clock from pin 11 of 32, are run through an AND gate and sent to the clock input of the divider. If a line feed cycle has just happened, and the FWD/BKWD line is low, the short pulse from pin 12 of 26 is added on to the normal clock pulses, causing the cursor to come one line earlier. If the FWD/BKWD line is high, the long pulse from pin 4 of 27 covers the first two normal clock pulses and the up pulse. Note that the up and backward pulses come even if the FWD/BKWD line is high, since they will be blocked out by the down and forward pulses anyway.

The divide by 25 chain consists of circuits 33 and 34, and its output is pin 11 of 34. This output is the clock input to a J-K flip flop, pin 1 of 32. When the output of the counter drops, the flip flop changes states, and pin 14 of 32 goes high. This flip flop is reset by the next character clock pulse, so pin 14 of 32 goes high for only one character width.

This line is called character position and goes to the chip enable of the character generator, which causes the generator's output to go high (display white on the screen). This is the cursor, and it goes to a blinker circuit before it gets to the character generator, which switches it off and on at the rate of about two times per second.

The character position line is ANDed with the update cycle and that pulse is sent to the page memory location that it is now at. The memory therefore gets changed during the update cycle, while the position of the cursor gets changed during the frame immediately following the update cycle.

To clear the display and move the cursor to the upper right hand position of the screen, the clear line is brought high. This brings the write line of the page memory high. It also brings the input to the memory bit 6 high. All keyboard data is gated to the memory inputs only when the update cycle is high, so the other five data inputs will be low. The ASCII code for a space is all bits low except for bit 6. So what will happen is that the memory will take the input data (space) and keep putting it into memory until the clear line goes low. If the clear line was high long enough to allow all of the memory positions to be accessed (1/60 second), the memory will be filled with blanks.

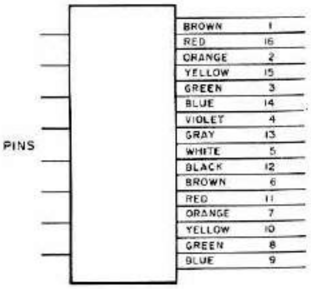


Fig. 9.

Now that the screen is blank, the cursor has to be moved to the first position of the page. To do this, both of the cursor divider chains are set to all ones; on the next clock pulse the divider's outputs will drop, displaying the cursor. A set/reset flip flop is set when clear goes high, and its output (pin 13 of 27) is used to preset the divider chain. Since the first clock pulse that gets to the divider must be at the beginning of the frame (so the counter will drop after the first clock pulse, or in the first character position), the preset line must go back to its normal state at the beginning of the frame. The preset line goes low the first time that the $\overline{T}$ clock goes low, after clear drops. Since the $\overline{T}$ clock goes low at the beginning of the frame, so does the preset pulse, and the dividers are now clocked.

Next Line Blanking

The next line blanking schematic can be found in Fig. 18.

The next line blanking circuit makes sure that, after you fill the screen and are about to start writing over old information, old information gets erased. It clears the screen line by line, as the cursor moves downward.

The lines get "blanked" under the following two conditions:

1. If the cursor is detected in the first character position of a line. In this case, the line that the cursor is on gets filled with blanks.

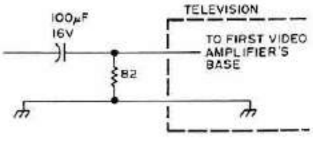


Fig. 10.

2. If a line feed is initiated. (The cursor moves straight down.) In this case, the line below the cursor's gets filled with blanks. Therefore, after the line feed is finished, the cursor is on the blank line.

To find whether or not the cursor is in the first position of any line, character position is ANDed with a signal that is high only in the first position of each line. This signal is pin 12 of 40, which is a J-K flip flop that gets set at the beginning of a line and reset after the first character position of the line is passed. Since the first $\overline{\text{load}}$ pulse lags behind the first line clock pulse by one character (as explained in the memory section), when both of these are high, and we are in the character area, we are also in the first position of a line.

As was stated earlier, this pulse is ANDed with character position, and if it is not "any cycle" or the frame right after it, it is used to set a flip flop (circuit 43). This flip flop's output (pin 13) is reset to zero the next time the K clock goes high (at the end of the scan line). This line goes on to bring the write line of the memory high, along with the bit 6 input, so that the memory gets filled with spaces. Since this line goes high at the beginning of a scan line and drops at the end of it, 40 locations of memory get filled with spaces. The memory gets written with spaces every cycle that the cursor is in position one.

In the second case, if an LF cycle occurs, it is ANDed with character position to set a flip flop (circuit 43). This will bring pin 4 of 43 high, and will allow the next flip flop (circuit 24) to toggle. This will happen the next time the N clock drops, or at the beginning of the next character line. This pulse is fed back to the first flip flop and resets it. The line is reset the next time that horizontal sync goes low. Therefore, the line goes high for almost one scan line, on the character line below the line where the cursor now is. (Remember that the cursor moves after an LF cycle drops.) This line also is fed to the write line of the page memories and to set bit 6 of the inputs high.

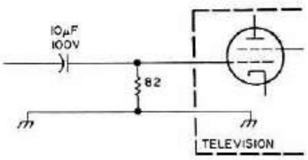


Fig. 11.

To turn the next line blanking feature off, jumper on the board the two pads marked NLB. This grounds pin 11 of 29, and doesn't allow a write pulse to go to the memory.

Power Supply

The power supply schematic can be found in Fig. 14.

The power supply has provisions for +5, -5, and -12 volts at 2, .15, and .5 Amperes, respectively. The +5 supply consists of a 12.6 V, 2 A, C.T. transformer whose output is rectified and filtered and sent to two 7805 (LM340T-5) regulators. The two regulator chips are mounted to the cabinet and are hooked up in parallel.

The -12 supply is obtained by rectifying and filtering a 28 V, .85 A, C.T. transformer and feeding it into a 7912 (LM320T-12) regulator chip. The output of this chip is -12, and that output is sent to the -5 regulator. This consists of a zener diode hooked to the base of an MJE370 transistor. The output is between -5 and -6 volts.

All outputs are filtered with .1 disc capacitors and 100 uF electrolytics to improve regulator response and to filter out noise.

Troubleshooting

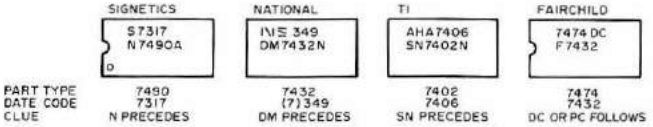


Fig. 12.

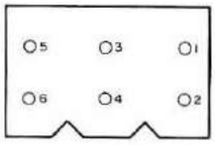


Fig. 13.

Power supply: When the board is finished, the power supply is not connected to the logic circuits. This prevents high voltage from ever reaching the integrated circuits, unless the power supply fails after it is initially tested.

To test the power supplies, the transformer is plugged into the board (through the molex connector) and the voltages are measured at the terminals marked +5, -5, and -12. If all of the voltages are correct (according to the table below), the three sets of pads can be jumpered, and the voltages measured again. If the voltages are still OK, the logic test procedure can begin. If not, you should start checking for solder shorts on the power supply lines.

	Max.	Min.
+5	4.75	5.25
-5	-4.80	-5.90
-12	-11.7	-12.3

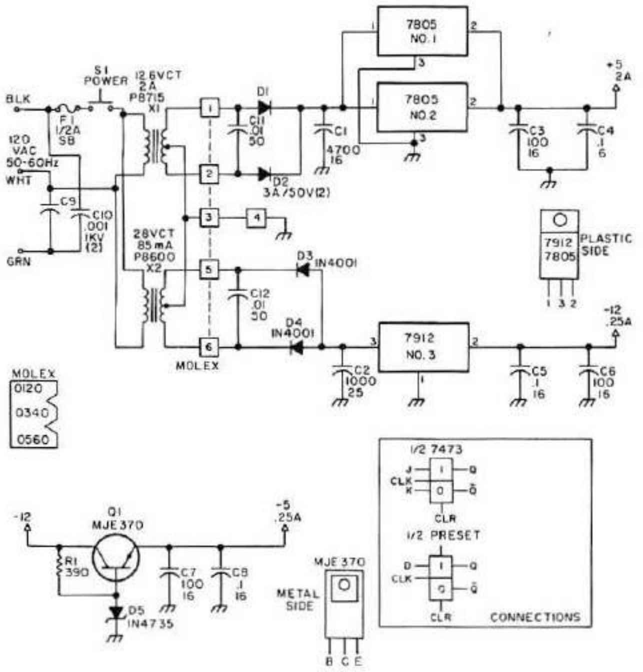


Fig. 14. Power supply.

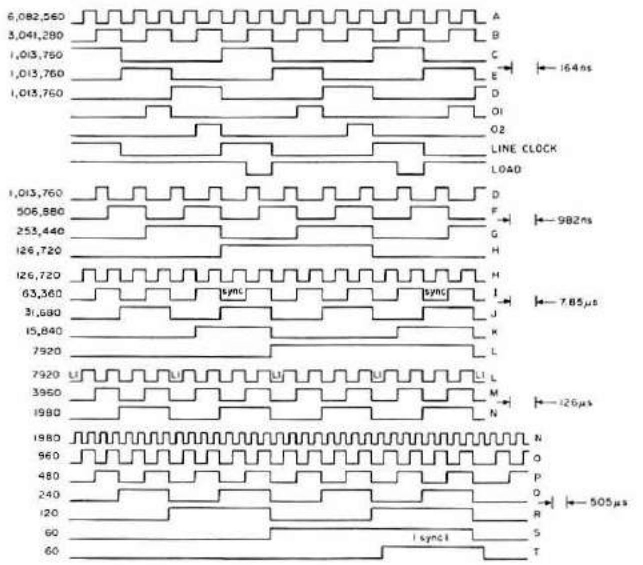


Fig. 15. Timing chain.

Logic testing: The 3320 display has a self-test feature which makes troubleshooting much easier. With this you can see any pulses that happen in the "visible" area of the frame. The visible area means any area of the frame that can be displayed on a television. Therefore, you can't see the sync pulses, and you must have these pulses to be able to display anything at all.

To use self-test, disconnect the jumper between the output shift register and pin 1 of 56 (between points D and E). This will take any video data that you may have off the screen. Next, connect pin 1 of the input connector to the signal line that you want to check. There should be a series of horizontal (for slower clocks) or vertical (for faster clocks) lines on the screen, where white is a logical one and black is a zero. To be sure that sync is there, and to be sure that the display is hooked to your television correctly, follow the above procedure and hook your jumper to a moderately fast clock (such as the G clock). In this case you should see 10 or 11 pairs of black and white vertical bars on the screen. If nothing is there, first check the connection to your television; if you're sure that's OK, you will need a scope to check the sync pulses and the output of the buffer transistor. After you get self-test working, go through any circuit that you think is bad and look at all of the signals. Please note that only TTL level signals can be displayed using self-test, so don't look at the output of the page memory's clock driver, or either of the negative supplies!

If the unit does not work, the first thing to check is the timing chain. The T clock should be 60 Hz (one horizontal line on the screen). If it is not there, start at the A clock and work your way through the chain. If the master clock is not oscillating, be sure that IC1 is a 7404 made by Texas Instruments. If it is not, it must be replaced with one that is. If the timing chain is all right, check the Ø1 and Ø2 clock pulses at pins 3 and 11 of 19, respectively. There should be 1024 small white dots on the television screen. If there are only 1000, check the divide by 24 circuit, ICs 21, 13, and 22. If there are dots all over the screen (many more than 1024), make sure that character area is high only for the area shown in Fig. 2, and that the divide by 24 circuit is working. L1 should be one horizontal scan line out of eight white, while load should be somewhat similar to the C, D, and E clocks, except the pulses will only appear during character area.

If everything seems to be OK in the timing section, check the cursor control. If a cursor appears on the screen, but won't move when pulsing the strobe line, first be sure that the jumper is in for positive or negative strobe! Then, see if there is a pulse at pin 5 of 36 that lasts exactly one frame (it will appear as a flash on the screen). If this pulse is not there, try clearing the unit by bringing the clear line to +5 volts. Because of the random states that logic

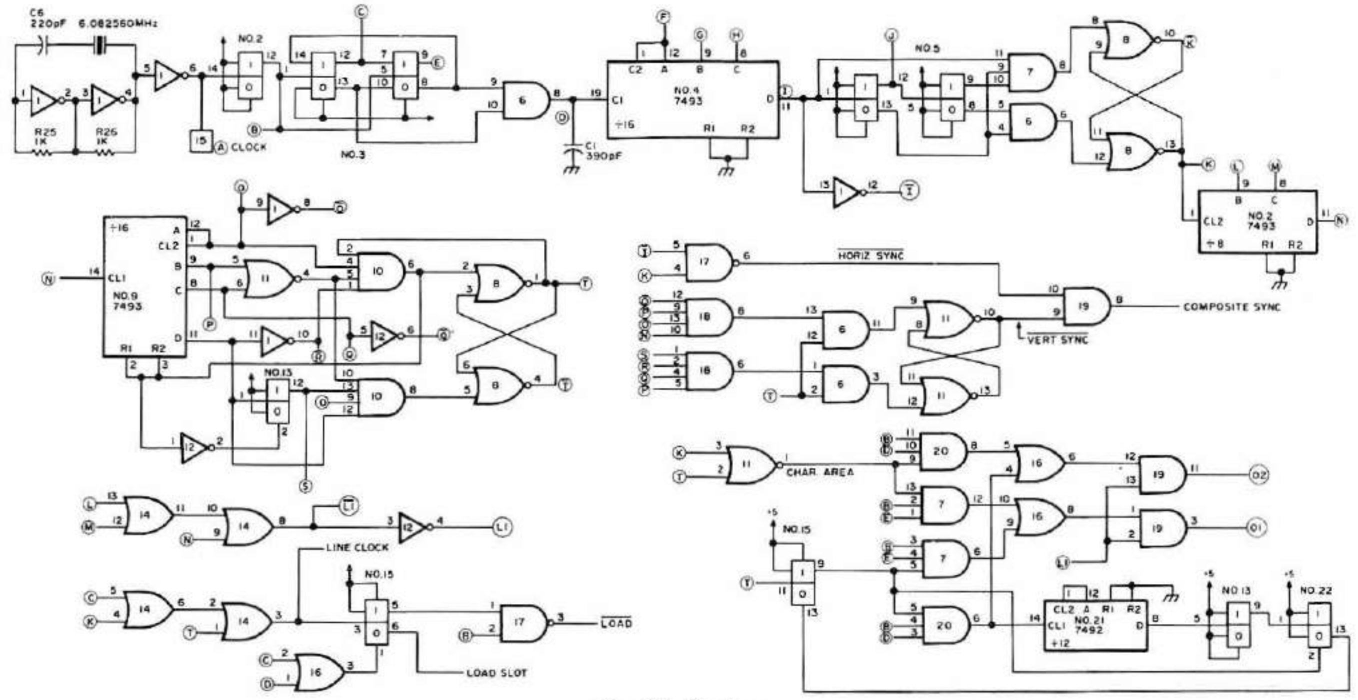


Fig. 16. Timing.

comes up in, some units must be cleared before data can be entered.

If a cursor doesn't appear, look for a character clock (1000 pulses on the screen) at pin 11 of 25. Be sure that this is clocking the flip flops and the divide by 10 (ICs 30 and 31). There should be 25 short normally high pulses per frame at pin 11 of 32. These 25 pulses are sent through an AND gate to clock the two divide by five circuits. If nothing is clocking, check the clear lines to the flip flops and the dividers. The output of the second divider (IC34) should cause flip flop IC32 to toggle once per frame. This flip flop will soon be reset by the next character clock pulse. This line is called character position, and is used to put the cursor on the screen and to bring the write line of the memory high during an update cycle.

If the timing and cursor sections both work, but the memory either won't enter data, won't fill with spaces when the clear line is brought high, or is creating random characters on the screen, the

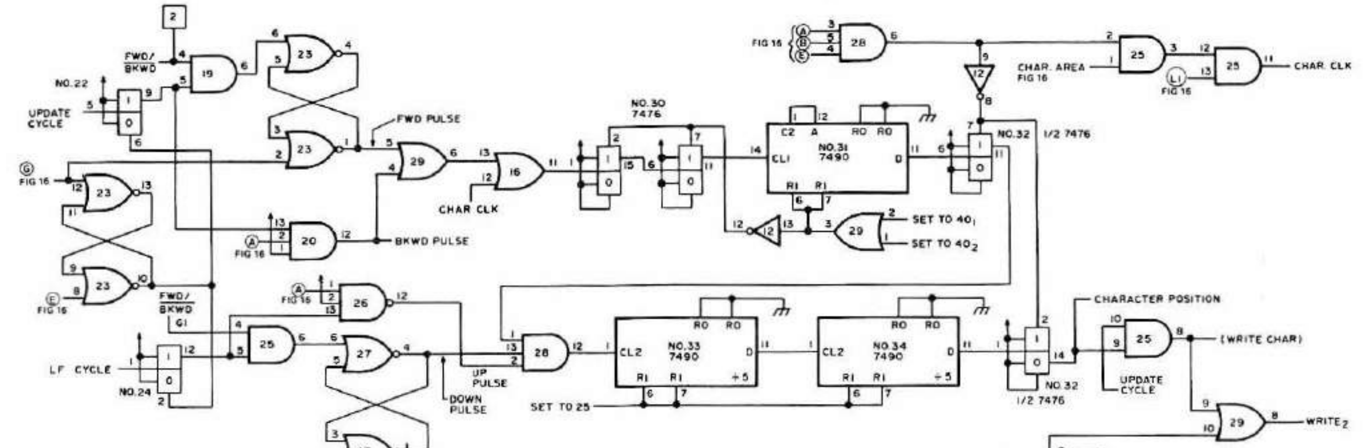


Fig. 17. Cursor control.

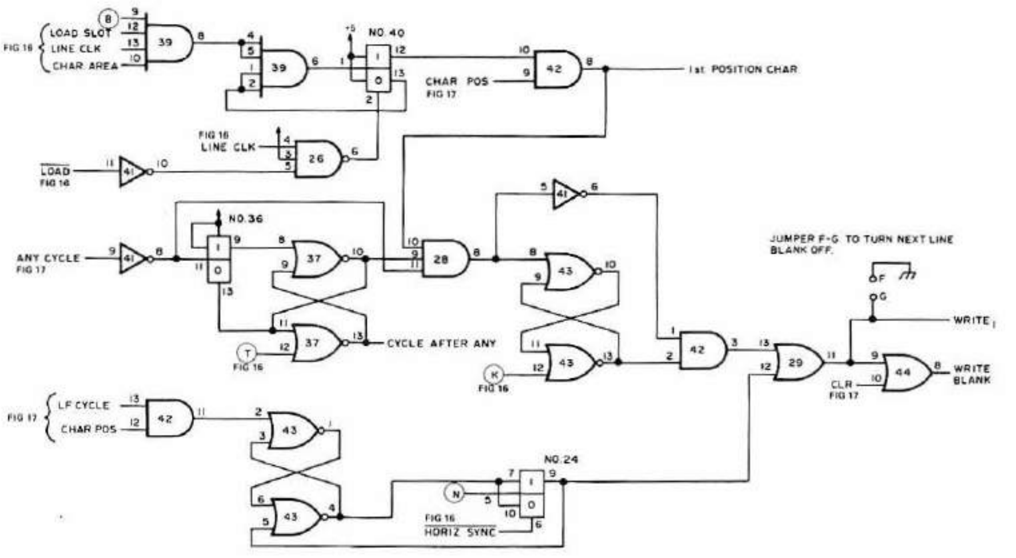


Fig. 18. Next line blanking.

memory section of the unit must be checked. First, if the screen has a series of vertical white lines on it, check pin 11 of 53 for a low. If it is not low (it should go high for a brief period to display the cursor), the character generator chip is not enabled and all ones will appear at its output. If this is not the problem, check all of the TTL level clocks such as L1, Ø1, Ø2, A clock, and the line clock. If these seem all right, check to be sure that the write line (pin 5 of circuits 46-51) goes high for a very short period during an update cycle. If it does not, check back through the circuit for the two write lines to see where they stop working. If everything is OK so far, you will need an oscilloscope to check the outputs of the clock drivers, circuit 45. There should be very little overshoot on these two clocks, and they must swing from about +5 to -12. *Do not look at these high level clocks with self-test. If this was done, circuit 56 must be replaced.* If the screen has a lot of characters that are shifting randomly, be sure that there are 1024 clock pulses at pins 11 and 3 of 19. If there are 1024 pulses per frame (on the screen), you must check the high level clock pulses from the clock driver *with an oscilloscope.*

If everything works except the next line blanking, be sure that you did not jumper the NLB pads on the circuit board, which will cause the next line blanking feature to be turned off. If the feature works with a line feed but not for a character in the first position, move the cursor to the first position of any line and check for a short pulse at pin 8 of 42. There will only be one pulse per frame. If it is not there, be sure that pin 2 of 40 is not always grounded, causing the flip flop to stay cleared, and that the same flip flop is being clocked at pin one. If that pulse is OK, see if it is getting through if it is during an update or line feed cycle (any cycle) or is the cycle after such. Pin 9 of 28 goes low for the cycle after any cycle. The pulse at pin 8 of 28 sets a set/reset flip flop, circuit 43, at pin 8, which brings the write line high. The line goes low when the K clock comes again. If the unit does not blank the next line after an LF cycle, see if the flip flop, circuit 43, is being set at pin 2, making pin 4 go high. If it is not, be sure that pin 5 is not always high, holding it reset. If pin 9 of 24 is always low, be sure that pin 10 of that circuit is not pegged low, keeping the flip flop cleared. The output of this flip flop also goes to set the write line high for one line.

If you find that a circuit is "bad," first check the following things:

1. That the power supply and ground are at acceptable levels on the chip.
2. That all of the input signals are within TTL specifications (low is less than .8 V; high is greater than 2.4 V).
3. That there are no unsoldered pins to the circuit, and no etch or solder shorts between a wire running to the chip and something else.

A large number of problems can be solved by checking these things before replacing an integrated circuit.

Operation

All connections to the keyboard and video monitor are through a DIP plug and a

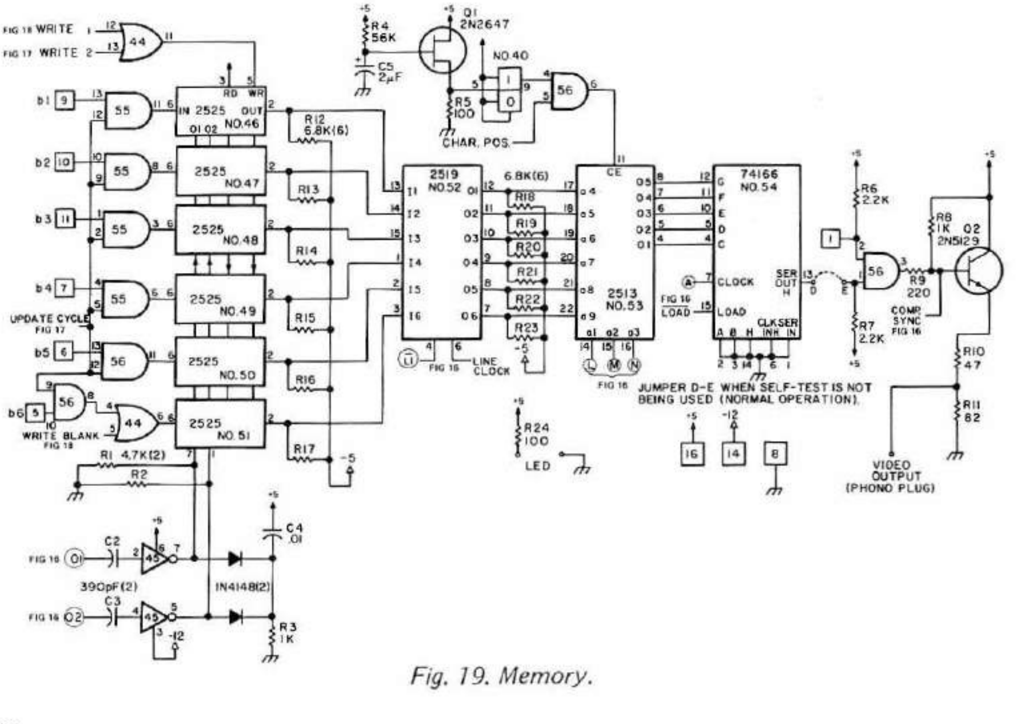
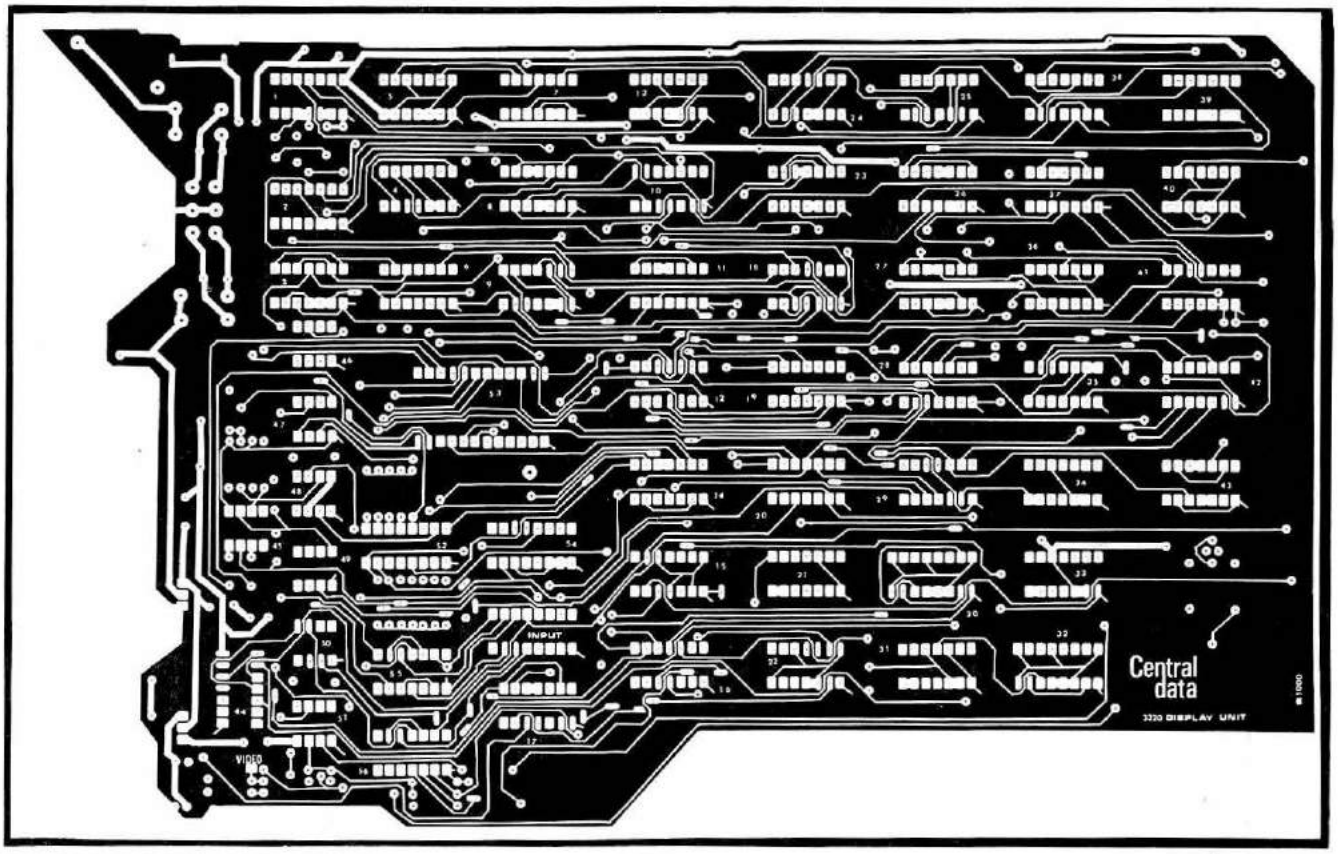


Fig. 19. Memory.

phono plug, respectively. The phono plug goes directly to the input of your monitor. It is the 2.25 V composite video signal whose waveform was given in Fig. 3.

The other connector consists of a 16 pin DIP socket soldered to the board, along with a cable that has a plug which will fit the socket. The cable's wires correspond to the pins on the plug as is shown in Fig. 9.

The signals that are available at the socket are listed in Fig. 19, the memory schematic. The strobe received signal is an output that acknowledges receipt of



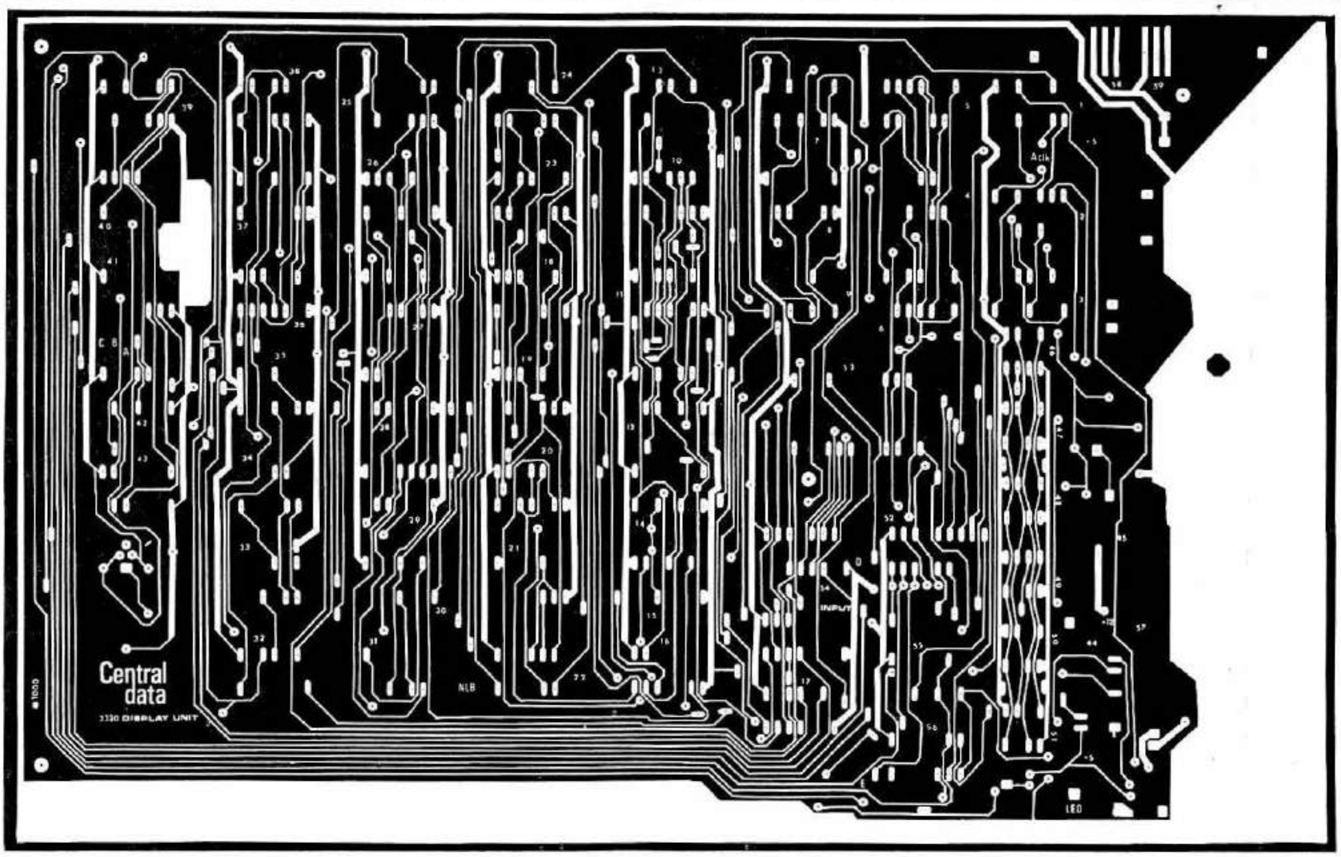
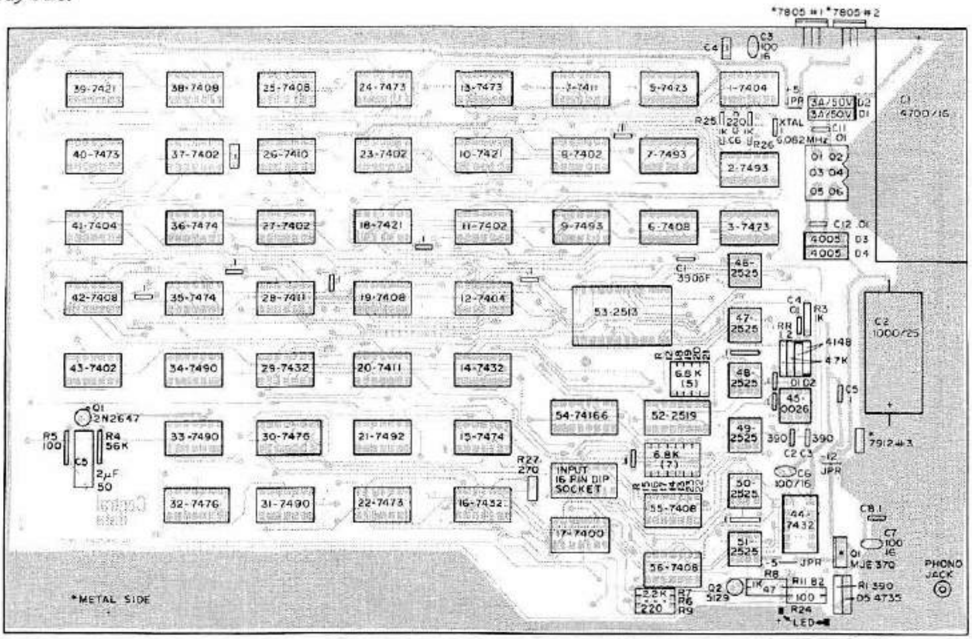


Fig. 20. PC board and component layout.



the strobe signal. It is a very short pulse that is used by the input options as well as some types of keyboards. Bringing CLEAR nigh will empty the screen and move the cursor to the upper left hand position. Sometimes, because the logic comes up in random states, the unit will have to be cleared before it will operate.

FWD/BKWD is an input that decides in which direction the cursor will move. If it is high, the cursor will move forward or down, while if it is low the cursor will move backward or up.

If you have an input option there is another DIP socket on the small option board which matches the one on the main board. The cable in this case had a DIP plug on both ends.

Modification of Your Television

WARNING: Do not attempt to use as a video monitor any television that has one side of the ac line connected to the chassis or to ground. To connect such a television will probably cause major damage to your display unit. The modification procedure is simply a matter of capacitively coupling the video output from your display unit to the first video amplifier stage of your television.

If you have a transistor television, use the circuit of Fig. 10 as a guide for the modification. If you have a tube type set, the circuit of Fig. 11 can be used.

With the tube type television, a higher voltage may be required to provide adequate contrast. A video amplifier can be inserted between the video output and the capacitor to solve this problem. With either set, there is a small possibility that it requires positive sync polarity. Since the data is transmitted with negative sync, a video inverter must be

Parts List

TTL Integrated Circuits

1	7400
6	7402
3	7404
7	7408
1	7410
3	7411
3	7421
4	7432
6	7473
3	7474
2	7476
3	7490
1	7492
3	7493
1	74166

MOS Integrated Circuits

1	MH0026CN
1	2513
1	2519
6	2525

Semiconductors

1	2N2646
1	2N5129
2	1N4148

Resistors (all ¼ Watt)

1	47
1	82
1	100
1	220
1	270
4	1k
2	2.2k
2	4.7k
12	6.8k
1	100k

Capacitors

1	220 pF disc
3	390 pF disc
1	.01 uF disc
13	.1 uF disc
1	2 uF/16 V, electrolytic
2	100 uF/16 V electrolytic

Miscellaneous Parts

5	Plastic standoffs
1	6.082560 MHz crystal
1	16 pin DIP socket
1	16 pin DIP plug with cable
1	phono jack
1	phono plug
1	Central Data B1000 circuit board

The following apply only if an on-board power supply is being built.

Semiconductors

2	7805 or LM340T-5
1	7912 or LM320T-12
1	MJE370
1	1N4735
2	1N4001
2	3 Ampere diodes

Capacitors

1	1000 uF/25 V electrolytic
1	4700 uF/16 V electrolytic
2	.01 disc
2	.001 disc (1kV)
2	.1 disc
1	100 uF/16 V electrolytic

Other Parts

1	390 Ohm, ¼ W resistor
1	1 Ampere fuse
1	fuse holder
1	line cord
1	strain relief for line cord
1	SPST toggle switch
1	Molex 03-09-1064
1	Molex 03-09-2062
6	Molex 02-09-1133
6	Molex 02-09-2143
1	Stancor P-8715 transformer
1	Stancor P-8600 transformer

Plus other misc. hardware

inserted between the display unit and the television.

Assembly: General Information

The circuit boards used in the 3320 display system are all double-sided with plated-through holes. Because of this you only have to solder the components on the bottom of the board. The plated-through holes make the connection to the top of the board. The components should be soldered with a low wattage (40-60 W), moderate temperature (approximately 750°) soldering iron. The iron should also be grounded when soldering the MOS integrated circuits. This is not absolutely necessary, however.

Be sure to mount all integrated circuits according to their pin one mark, not their lettering. Occasionally a circuit is lettered upside down. Most ICs are printed with both a part number and a date code. The date code consists of the year and the week that the circuit was produced. Example: January 30, 1977=7705.

The placement of the part number and date code for most circuits is shown in Fig. 12.

Also, be very careful to prevent solder shorts. This type of problem is very difficult to find later, so if it is done with care now it will save you a lot of time when troubleshooting.

Note that all pin one markers on the integrated circuits are towards the lower left hand corner of the board, except for circuit 44, which is toward the lower right hand corner. IC1 must be a Texas Instruments circuit; otherwise you are not assured that your master clock will oscillate. A square pad on the top of the board indicates the cathode (banded) end of diodes and the positive end of electrolytic capacitors.

Be sure to jumper the two terminals marked "A clk" near circuit 1. If you ever want to run your display off an external clock, break the jumper and connect the external clock to the lower pad. Also jumper the correct pads between circuits 41 and 42 corresponding to the polarity of your strobe pulse. If you are running off an external power supply, wire your power supply to the circuit side of the three sets of pads marked +5, -5, and -12. Also jumper pads D and E to connect the shift register to the buffer transistor.

A phono jack is installed in the ¼ inch hole, and a jumper is made between its center and the pad on the bottom of the board marked VIDEO. Connection to your monitor can be made using a phono plug and some RG-174U cable.

If you are going to use the on-board power supply, follow the parts placement diagram shown in Fig. 20. Be sure to mount the 7912 and the MJE370 transistor on heat sinks. The two 7805 regulators must be mounted at right angles to the board, and must be attached to a heat sink at least 4" x 9" x 1/16", if aluminum. To mount the regulators to the board, bend the leads (where they taper off) at a 90° angle. Then put some solder on each of the leads, along with the pads on the circuit board where they will be mounted. Then set one regulator on the circuit board and, one by one, heat the leads so that the solder melts and attaches the lead to the board.

Solder the transformer leads to the off-board molex connector using the configuration shown in Fig. 13. Two .001 disc capacitors should be placed across the power line to prevent high frequency noise from leaving the unit through the line. ■

I/O EDITORIAL

POSTAL DISASTER

With first class mail looking to go to 17¢ by next year and to around 25¢ in a couple of years beyond that, some viable alternative is a must. The post office is already spending money (though not in *73*) to try and get people to send letters as a result of the drastic drop in mail use brought on by the 13¢ letter. There is also talk of a three day postal week.

Of course it is always possible that pressure on Congress will get so great that the postal monopoly will be broken. Right now it is a federal crime to compete with the post office. The mail could be carried at a good profit by private industry, if it were permitted. Industry could get the postal system back on its feet if permitted by the bureaucracy and the postal unions, something that doesn't seem even remotely likely.

Last year the post office rang up an $825 million deficit and it is expected to be almost double that by the June 30th end of the current fiscal year.

Every time I pause to think about our having to send a piece of paper from New Hampshire to California in order to communicate a message it makes me angry. It is ridiculous in these times. Letters can be sent by facsimile, by teletype, or some computer code and sent via wire, satellite or microwave. Western Union screwed up . . . if they had been looking ahead they would have gotten aboard some really inexpensive message system. Ma Bell (you probably noticed the page 3 mention, in May, of the wind up of her suit against *73* . . . *73* admitted no fault) could have worked up a good system using Bell wires and communications systems.

The soon to arrive inexpensive small office and home computer surely will blow off the lid. I'm still talking with the salesmen for the large computer firms and they apparently are still completely unaware of the revolution that is brewing in small computer systems. They say, oh yes, the microcomputer will cut the cost of the CPU (central processing unit), but there is no way that the cost of the peripherals is going to come down substantially . . . and after all, the cost of the CPU is not very significant for most systems, so therefore the cost of computer systems will stay up there in the $50,000 plus range for a long, long time.

Baloney.

Please don't tell the ivory tower boys about what is happening . . . about what is coming. We don't need IBM in there . . . or DEC. Let them keep working on their $100,000 systems and selling them the way they are right now. This will make it possible for a whole new bunch of companies to grow into the new field rather than have the old ones just keep getting bigger and bigger.

Let's take a look at the peripherals. One thing we need a lot of with any system are the CRTs . . . call them video typewriters or television typewriters (TVTs) . . . these are the primary input/output devices we will be using. Today you can buy a real dumb TVT for maybe $1500. The more sophisticated ones run double to triple that. I envision a CRT with a keyboard and a lot of smarts selling for under $200. It will have a microcomputer chip in it and be able to do a lot . . . play games . . . let you type letters and edit them . . . store a lot of form letters . . . do bookkeeping . . . keep all sorts of files . . . and a whole lot of things we haven't thought of yet. It will need some sort of file cabinet memory storage . . . probably not invented yet.

There are some storage devices in the works. Right now it costs about $5000 for a floppy disk system . . . that should drop to about $1500 in a few weeks when Diablo's new dual $1000 floppy drive is released. Others are in the works for $600. Somewhat slower tape systems will be able to hold enormous amounts of data . . . and at prices which will make DEC salesmen shake their heads in disbelief.

Speaking of Bell . . . the phone company got itself into the Blue Box situation because some engineers ignored a lot of the savvy of other engineers who warned them that using the same lines for signaling and customers was a bum deal. In the long run it is costing Bell heavily for trying to use one pair for everything. A relatively few signaling lines could switch vast numbers of communications lines . . . and there would be no problem with customers sending signals to bollix the charge meters.

Funny thing about the series we published on the phone circuits . . . until Bell made a big deal about it no one seemed to pay any attention. Bell could hardly have gone more out of its way to make sure that the article was noticed . . . coast-to-coast publicity on the radio, television and in the press! They sure brought a load of inquiries on *73* for copies of the magazine or the articles . . . we probably could have sold several thousand copies of the material if we'd been interested in helping people to defeat the phone system . . . but we just turned down all the requests.

Now, after paying several thousand dollars in legal fees, I don't think I should have been quite as solicitous of Bell's interests. On the other hand, after reading the underground telephone newsletter, I am convinced that Bell is able to locate Blue Boxes fairly fast these days and the circuits, even if people discovered how to use them . . . which was not explained in the articles . . . you had to know that or read it elsewhere . . . they would have gotten into trouble.

Using A Bargain Surplus Keyboard

by
Cole Ellsworth W6OXP
10461 Dewey Dr.
Garden Grove CA 92640

I recently became the owner of an ASCII (American Standard Code for Information Interchange) encoded four-row keyboard purchased from a local parts emporium. Of relatively recent surplus vintage, it had originally been used in a computer terminal. The keyboard electronics consisted principally of a 40-pin LSI encoder chip similar to the General Instrument Corporation AY-5-2376. Several 7400 series chips composed the remainder of the circuit. Parallel format outputs included the 7-bit ASCII code, plus a parity bit, and the key-pressed signal (keyboard strobe). In addition, two key-switch non-encoded functions were available: "REPEAT" and "INT" (similar to WRU). All outputs were TTL-compatible positive true logic levels, although each of the eight data bit outputs was capable of driving only one TTL load.

The reason for acquiring the keyboard in the first place was to incorporate it in a computer terminal with video display to provide a man-microprocessor communications link. However, other applications came to mind, such as converting the 8-level ASCII code to Baudot 5-level code for amateur band use. And, should the FCC see fit to permit 8-level codes on the amateur frequencies, a proper interface would make such operation easy to accomplish. It became apparent that a number of keyboard support functions such as character counter, EOL indicator, repeat function generation, data output buffering, and keyboard strobe control would be required to provide maximum versatility in the aforementioned applications.

After an analysis of the keyboard support requirements and a projection of probable usage of the keyboard in varied applications, a logic diagram evolved that met all the requirements for my particular keyboard. It should be noted that keyboards, like canned soup, come in many varieties. Some designs have data outputs that are negative true TTL levels (mark = low level). Some have mixed outputs where the strobe is positive logic and the data bits are negative logic. Some recent designs have on-board LSI encoders that have a built-in repeat function (the strobe signal pulses at a ten Hz rate when a character key is held down more than about ½ second) while others do not even have a

repeat key. So as to make the support logic as versatile as possible, a number of options were provided in the final design and the result was named the Keyboard Interface – version 1 (KBI-1). Fig. 1 illustrates the interface unit logic diagram and connection to the keyboard. Fig. 2 shows the Digital Read Out (DRO-1) logic diagram which includes the EOL decoder.

Features

1. Provides buffered (up to 30 TTL loads) positive true data outputs (U3, U4) for the eight data bits with either positive true or negative true inputs.

2. Provides for strobed (3-state) outputs to allow connection to standard data bus, or normal 2-state outputs by means of jumper JM5.

3. Provides repeat function generator if desired (JM6).

4. Allows strobe and repeat functions from either positive or negative logic keyboard outputs (JM1, JM2, JM3, JM4).

5. Provides four variations in strobe pulse outputs:

a. Negative-going strobe pulse remaining at a low level until key is released (strap JM7 A to B).

b. Positive-going strobe pulse remaining at a high level until key is released (strap JM7 A to C).

c. Negative-going strobe pulse with variable delay and variable pulse width (strap JM7 A to D).

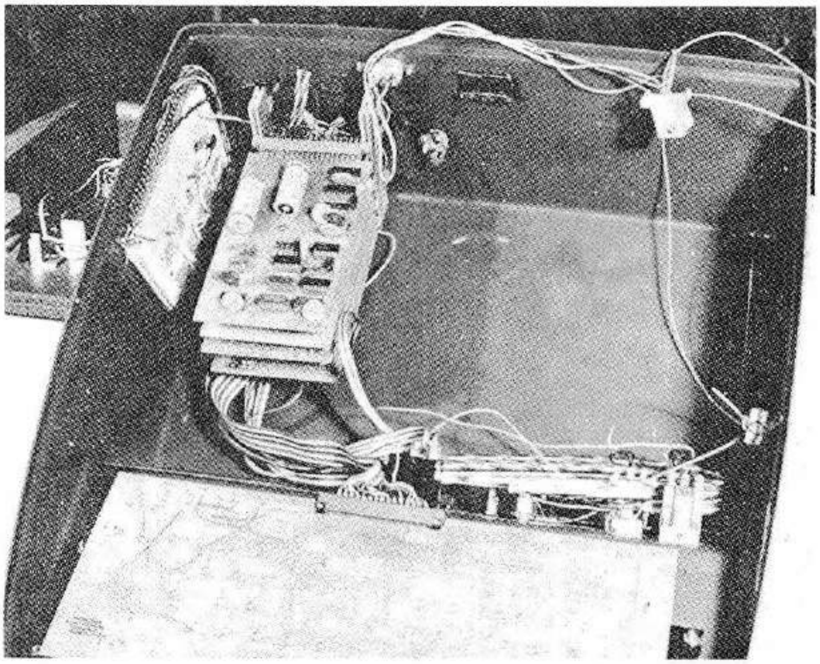
d. Same as letter c except positive-going strobe pulse (JM7 A to E).

6. Provides a character counter and LED display that counts only the printing ASCII characters plus space bar. Counter is reset to ØØ on receipt of ASCII LINE FEED function.

7. Provides End-Of-Line (EOL) indication at any desired character count. EOL indicator is also reset by the LINE FEED function.

Circuit Notes

For interface to keyboards with positive true logic outputs, U3 and U4 should be the 74367 non-inverting 3-state buffers. In this



Inside view of console showing method of mounting and interconnecting the five PCBs and keyboard. The PCB on the far left is a catch-all for peripheral drivers and other accessory functions.

case U3A and U4A are jumpered as follows: U3A – JM1 open, JM4 A to B; U4A – JM2 open, JM3 A to B. Note that in this configuration all keyboard data outputs must be positive logic including the strobe and repeat signals. There is no provision for handling negative logic strobe and repeat signals when data bits are positive logic. If required, outboard inverters could be used in this case.

For interface to keyboards with negative logic data outputs and *negative or positive* logic strobe and repeat signals, U3 and U4 are 74368 inverting buffers. Pinout and control levels are identical with the 74367. For a negative logic strobe from the keyboard, U4A JM2 is open and JM3 A to B. For positive logic strobe, JM2 is jumpered and JM3 A to C. Similar jumper conditions apply to U3A for negative or positive logic REPEAT signals.

If the keyboard has on-board repeat character function, then U1, U2A, and U3A are not required and may be disabled by connecting JM6 A to C (in this case U1 and associated capacitors and resistors need not be installed). If keyboard has the repeat function key but no repeat oscillator, connect JM6 A to B for a ten Hz pulsing of the keyboard strobe.

If normal non-strobed bipolar data outputs are desired (no high impedance third state), connect JM5 A to C. If 3-state data outputs are desired (U3 and U4 pass data only during presence of keyboard strobe pulse and outputs revert to a high impedance state when strobe pulse is not present), connect JM5 A to B. For initial tests of the KBI-1, JM5 should be strapped A to C.

The four variations in strobe signal output are selectable by JM7. Connect JM7 A to B for negative-going strobe staying low until key is released, JM7 A to C for inverse (positive-going strobe staying high until key is released). JM7 A to D and A to E select negative or positive strobe pulses as required. U14A R5 and C4 are selected to provide the

References

[1] Electronic Development, Inc., PO Box 951, Salem OR 97308.

[2] Hoff, Irvin M., "The Mainline UT-4," RTTY Journal, March, 1975, p. 4.

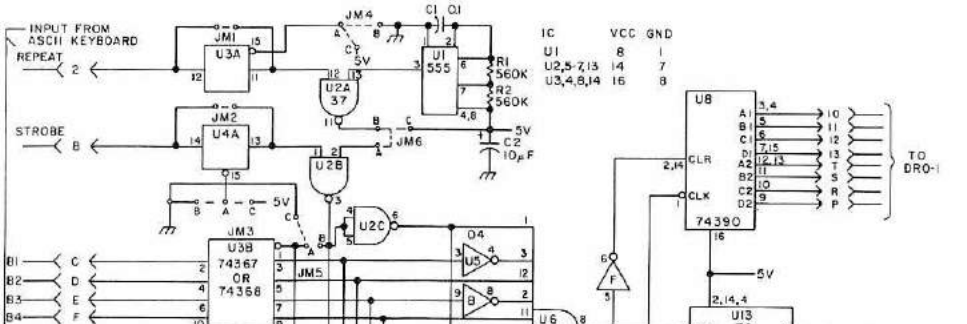


Fig. 1. Interface unit logic diagram and connection to keyboard.

required strobe delay, and U14B R6 and C5 are selected to provide the desired pulse width.

Thus, it is apparent that while the KBI-1 will provide interface for the majority of keyboard logic configurations, all possible permutations are not available. The KBI-1 is *not* directly compatible with non-coded keyboards, keyboards coded for IBM codes such as EBCDIC or SELECTRIC, or Keypunch (Hollerith) codes. In other words, before you consider use of the KBI-1, your keyboard must meet three conditions: 1. Outputs must be in a 7-bit parallel ASCII encoded format with or without parity bit. 2. Outputs must be TTL-compatible. 3. Must have a strobe (keypressed) signal output. The character counter section of the KBI-1 provides a two-digit display (ØØ to 99) of the number of printing characters (and space bar) generated. A portion of this circuit is located on the DRO-1 board (see Fig. 2). It will not count control functions. The display is reset to ØØ whenever LINE FEED key is pressed. EOL indication is provided by lamp DS-1. This circuit operates by detecting a preset number determined by strapping outputs of U11 and U12 to the inputs of U7D. Decimal thumbwheel switches (S1, S2) may be used for convenience in changing the set point if desired. Otherwise, straps are run from U7D inputs to the desired outputs on U11 and U12. At the preset count, the EOL indicator illuminates and remains illuminated until LINE FEED key is depressed.

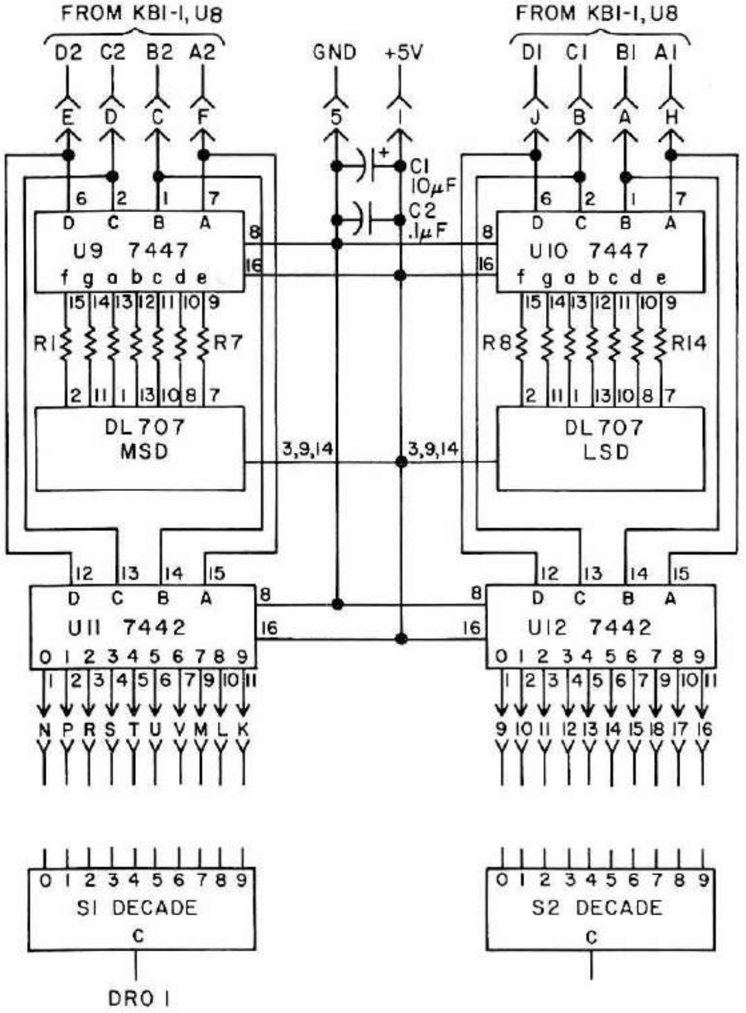
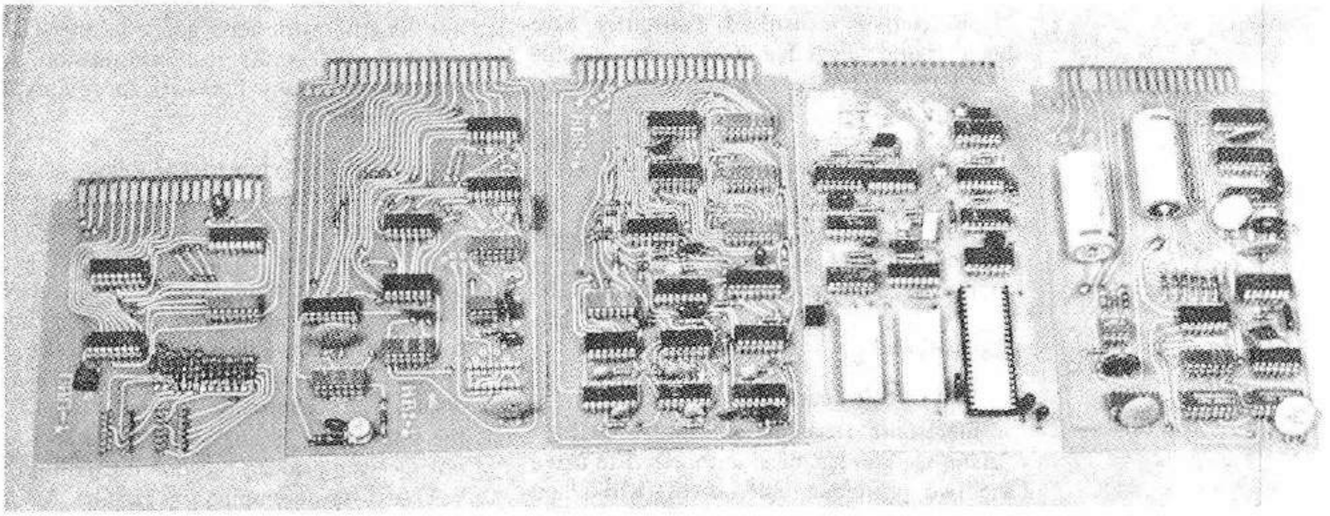


Fig. 2. Digital Read Out logic diagram, including EOL decoder.

Construction

Circuit layout and wiring is not critical provided normal rules of TTL logic are followed. Printed circuit boards greatly facilitate and speed construction and are recommended in particular to those who are not familiar with digital logic hardware. New, high quality components should be used to reduce or eliminate debugging problems. Surplus or reclaimed components may be used if you have the proper facilities



The five printed circuit boards from left to right: DRO-1 character counter and display driver (short board), KBI-1 interface, ABC-1 ASCII to Baudot converter, UT-4 i-f main board, UT-4 i-f auxiliary board.

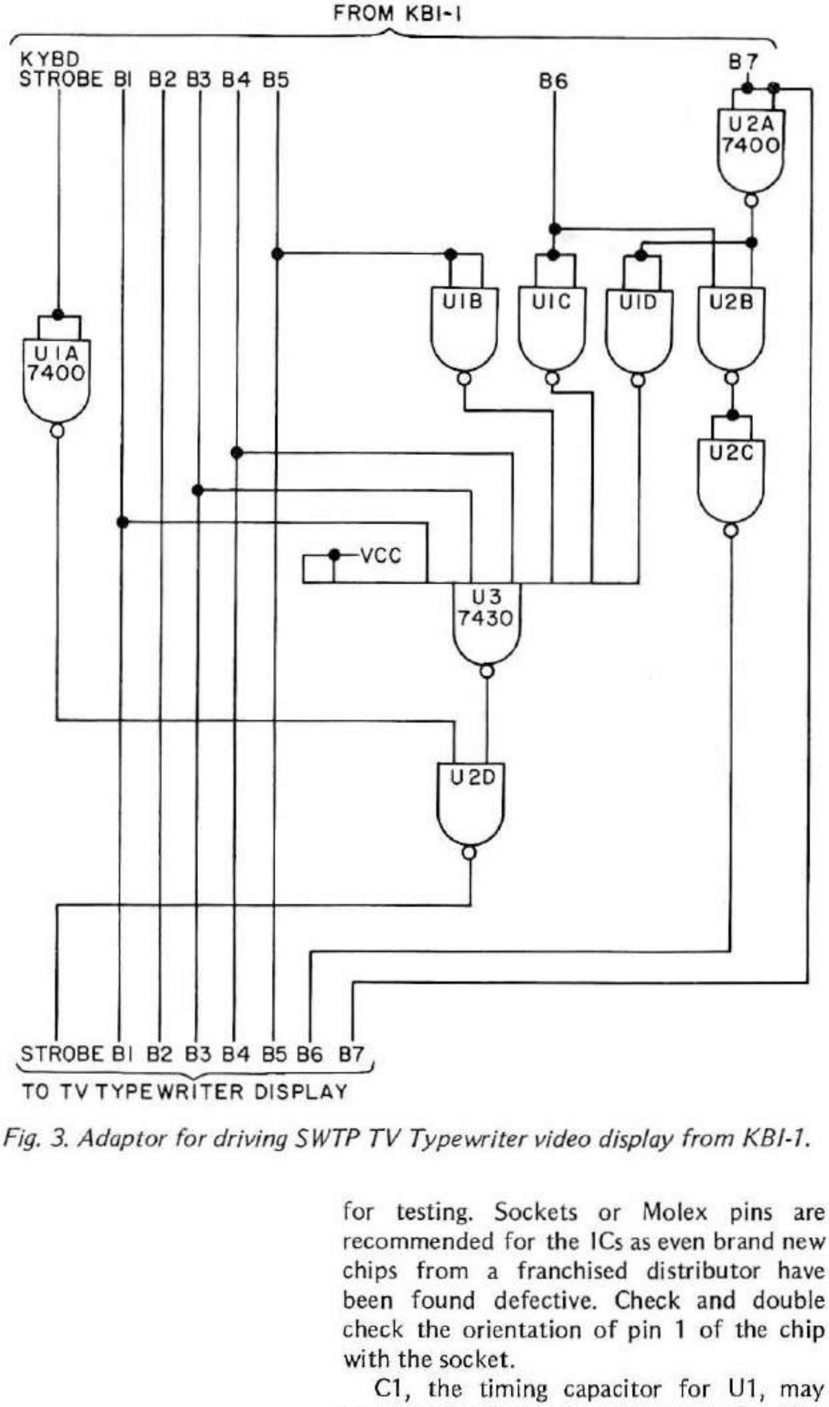


Fig. 3. Adaptor for driving SWTP TV Typewriter video display from KBI-1.

for testing. Sockets or Molex pins are recommended for the ICs as even brand new chips from a franchised distributor have been found defective. Check and double check the orientation of pin 1 of the chip with the socket.

C1, the timing capacitor for U1, may require selection to keep the repeat function rate at ten Hertz or less. R1 and R2 should be 5% ¼ Watt. R4 is required for the recommended EOL lamp and +12 volt supply. R4 may be replaced by a jumper if a 5 volt lamp and +5 volts are used in this application.

Printed Circuit Boards

Electronic Development Inc. (EDI)[1] of Salem, Oregon has been authorized to make the two printed circuit boards (KBI-1 and DRO-1) available for this project. No other sources of PCBs are expected to be available. PC boards only, or complete kits may be obtained. See EDI ads in *73* for description and availability. The boards are high quality epoxy glass, double-sided with plated-through holes and fit standard .156 inch spacing, 18-position double readout edge connectors. The edge connector references on the logic diagrams are identical with the EDI PCB edge connections.

Troubleshooting

At the time of this writing, three persons have built the KBI-1 and three different ASCII encoded keyboards have been used. These were surplus units manufactured by Clare-Pendar, Microswitch and Tektronix. Various problems were encountered in check-out including poor solder joints, missed solder points, trace to trace solder bridges, overloaded power supplies, and in several instances defective ICs were found. In each case a carefully thought out, logical approach to troubleshooting pinpointed the problem area. While a good triggered oscilloscope is a most useful adjunct to logic circuit troubleshooting, it should be pointed out that Peter K6SRG debugged and checked out early prototypes of the KBI-1, the ABC-1 ASCII to Baudot Converter, and UT-4 i-f circuits using nothing but a VOM (and an ice cube to find a temperature sensitive 74390!).

The KBI-1 was designed to be located in the immediate vicinity of the ASCII keyboard. Leads from the keyboard data outputs to the KBI-1 should not exceed 20 inches. Lines of greater length will cause ringing, especially on the keyboard strobe pulse. Ringing on the strobe pulse will cause multiple outputs from a single keystroke. Jim WA7ARI used a six foot cable between his keyboard and the KBI-1 but it was necessary to use type 75188 line drivers at the keyboard and 75189 line receivers at the KBI-1 inputs to prevent ringing. The same line length restrictions apply to the KBI-1 outputs. You must use transmission line techniques for any line over approximately 20 inches.

KBI-1 with TV Typewriter Display

Many RTTY enthusiasts have built the Southwest Technical Products Corp. (SWTP) "TV Typewriter" video display. Fig. 3 illustrates a special adaptor used between the KBI-1 output and the TV Typewriter input so that the video display has a CR/LF on receipt of LF instead of CR. The video display will still provide automatic CR/LF at the end of its 32 character line if an LF is not received prior to this point. Credit is due WA7ARI for developing this circuit.

Acknowledgements

Recognition and appreciation of their contributions, comments and encouragement are due members of the 3612.500 autostart net, in particular W6FFC, W6GQC, K6SRG, WB6WPX and WA7ARI. Thanks is also due those members of the net whose patience was sorely tried during the development period. Others who contributed comments and suggestions included K2SMN, WA5NYY and KL7HOH. The author extends his apologies to anyone whose contributions were overlooked.

KBI-1 Parts List	
U1	555
U2	7437
U3, 4	74367 or 74368 (See text)
U5	7404
U6	7430
U7	7402
U8	74390
U13	7474
U14	74221
C1, 6-8	0.1 uF
C2, 3	10 uF, 16 volts
C4, 5	selected, see text
R1, 2	560k 5%
R3	470
R4	75 ½ W
EOL lamp	6 V/75 mA

DRO-1 Parts List	
U9, 10	7447
U11, 12	7442
Display DL707 or equiv.	
R1-14	270 Ohm
C1	10 mF, 16 V
C2	0.1 mF
S1, 2	Decade thumbwheel switches

ASCII to Baudot Conversion

A forthcoming article will describe an ASCII to Baudot converter (ABC-1) that converts ASCII data from the KBI-1 to parallel format Baudot 5-level code and outputs this data to a FIFO/UART combination such as the UT-4[2]. ■

I/O EDITORIAL

COMPUTER CONVENTION

The MITS World Altair Computer Convention in Albuquerque came off very well with some 500 plus computer hobbyists attending. The convention was run in the Airport Marina hotel, just around the corner from the new MITS plant.

Since only Altair computers were being shown, there were not a lot of exhibits . . . but those which were there were of interest. Probably the most exciting was a ham station (WA8VNP) set up and operating in the RTTY contest (which was coincidentally on that weekend). The whole works was being displayed on a CRT with one area for the received copy, one for the copy being typed to be sent, another for checking for duplicates, and a fourth for logging. A built-in clock added the time to each exchange.

All the operator had to do was type in the call of the station to be worked and the report . . . the computer then checked to see if that station had yet been worked on this band . . . if so, it typed out "Dup." If not, it would go ahead with the contact when given the word to start.

Another system was showing a biorhythm display. It would ask your birthday, how many days you would like shown, and starting with what date . . . from there on it would plot your emotional, intellectual and physical curves, showing you why things went so wrong or perhaps what you might expect next week.

Cromemco's TV dazzler was one of those gee-I've-gotta-have things. I'm not sure what it can do, but it was impressive in its ability to show colors on a big color TV set. I suspect that someone into art could work up programs which would be real knockouts.

In view of some past hostility to firms making Altair-compatible boards, it was surprising and comforting to see one competitor there with a display of their boards.

The day and a half of the convention was all too little for all the talking and seeing to be done.

And hams? A substantial number of those who came had HTs sticking out of a pocket!

I finally found out what MITS stands for . . . seems they got started in 1969 making telemetry kits . . . it is Micro Instrument and Telemetry Systems. In 1970 they got into calculators and by 1974 had taken a terrible beating as calculator prices plummeted. They were just about to give up when they came up with the Altair 8800 computer – and the rest is history.

MY BIG SPEECH

Since the other editors in the field had been asked to give a talk, I came prepared, too . . . hoping that perhaps I had just been overlooked. It turned out that I apparently was not in the same league as the others, so I still have my notes.

The basic thrust of my proposed talk had to do with the almost unlimited opportunity that microcomputers afford. I doubt that there has ever been a time in the past when it was possible to see so clearly the opening of a whole new field . . . a multi-billion dollar field. Flying pioneers knew they had something, but they had no hint of the magnitude. You have to be fairly old now to remember the almost universal reservations which greeted television in its early years. My folks were really upset when I went out in 1948 and spent $250 for a ten inch black and white set . . . a waste of money . . . we'll never watch it . . . etc. We watched it . . . including wrestling, westerns, everything.

The hobby computing field is fantastic. It isn't very big as yet . . . in fact, it may never be very large . . . but it is going to have a profound effect on the history of the world. The computer hobbyist is going to pay for the technical development needed to build small computer systems which are needed by small businesses, schools and homes.

You can today buy a computer system for about $5000 that would only a couple years ago (or less) cost you around $50,000 . . . and this is only the very beginning. Let's look ahead a little . . . how long will it be before we have the $299 smart terminal? It will have a black and white TV monitor (they run about $75 new these days) . . . a 16K RAM memory for programs and storage ($20?) . . . a ROM with text editing and word processing programs . . . a printed circuit board keyboard like the ones now used for tone pads on HTs ($5?) . . . a video display chip ($5) . . . keyboard/ASCII chip ($3) . . . etc.

What office desk will be able to be without a desktop terminal? It will be used to type all letters, will work fine for form letters, will act as a file cabinet or card file, calculator, Teletype via phone lines, handle all the bookkeeping and accounting, send invoices, print out labels, inventory, etc. Each office will probably need some hard copy device . . . either a line printer or a photo copier to read from the tube. They will need some data bank . . . tape, disk . . . or something not yet invented . . . perhaps like a video disk system.

Every worker in an office will need a terminal . . . every retail store will need one to keep track of sales, inventory, costs, etc. Will the computer terminal be as common on school desks of the future as the inkwell was when I went to school?

And how about the home? The terminal here could be used for an almost infinite number of applications . . . security . . . heating . . . grass watering . . . message center . . . letters . . . banking . . . ordering from local super market for delivery by electric cart . . . ordering from Sears, etc. . . . any school course of instruction . . . games . . . and etc^{n}.

I see these markets as inevitable, and I believe that they will evolve from the present computer hobby systems . . . with hobbyists both paying the freight for this development and participating in it. I also see this as a very good way for things to go.

Let's suppose that IBM or DEC had the vision to come out with a $299 computer, complete with CRT, keyboard, some ROM programs, RAM memory and the ability to interface with larger memory, other terminals, line printers, etc. It just might happen that their present system of selling would stick . . . factory to customer via local factory reps and service centers. That would deal us out of the big ball of wax.

But the present system, which is developing with small manufacturers, local computer stores for sales, programming and service, affords a much more widely based market . . . one that can accommodate many manufacturers and still keep prices down. The pricing schedule of the big computer firms is a tough bone, with many items being marked up 100% when sold by other than the manufacturer. A $1500 printer then sells for $3000!

Computer stores will be much like hi-fi stores, carrying equipment of many manufacturers and mating compatible gear. The old OEM price schedule will hold for dealers and the markups will be more like 35% than 100%. Maybe even less.

Your guess is as good as mine as to how many terminals will be sold to handle some 44 million school children . . . 80 million homes . . . and perhaps 60 million workers who might need one. I'd say that it will be a large market . . . maybe 25 to 50 million terminals per year . . . plus peripherals such as memory, printers, etc. That would certainly come to 10 to 30 billion per year in sales. Not bad for a market that doesn't even exist today!

As long as the big computer firms don't notice this market, it will be wide open to the newcomer . . . and that means the computer hobbyist has a big advantage. Few others have the range of experience needed to get this

Continued on page 87

Stanley P. Levy WB6SQU
P.O. Box 961
Temple City CA 91780

A Morse to RTTY Converter

-- using a microprocessor

The advent of powerful, inexpensive microprocessors has made the goal of a good Morse code to TTY translator a practical reality. Only seven chips are needed to implement this Morse code to Teletype translator which is self-adaptive to code speed and spacing over a wide range – without adjustment. Either ASCII or Baudot output is available by changing the programming ROM. Serial output is provided which can be adjusted from 60 to 100 wpm.

This translator will accept a dot/dash ratio from 1:2 to 1:4 (1:3 is nominal for International Morse Code) with any ratio of letter speed to spacing (i.e., 25 wpm characters spaced out for 10 wpm) because mark and space timing are evaluated separately. A speed variation of ±20% per character can be followed without error. Larger variations require from three to ten characters at the new speed to regain lock. Word spacing is also provided, and carriage return is adjusted to avoid breaking up words at the end of lines.

Due to its simplicity, this translator can be assembled in only an hour or two. Circuit boards and components are available, as well as preprogrammed ROMs and complete programming information (see Parts List).

This machine is based on the MOS Technology MCS 6502 microprocessor. This device offers several features which make it attractive for this application: low cost, fast cycle time, on-chip clock oscillator, and single +5 volt power supply. Bus organization is the same as the M6800 and offers simple memory-I/O interface with a minimum of external parts.

The entire system includes a CPU (the microprocessor), a 512 x 8 ROM containing the program, a 128 x 8 RAM, an interrupt timer, and three TTL chips which provide a 1-bit input port, a 1-bit output port, and an interrupt control flip flop. No UART or other parallel to serial converter is needed since this function is provided by the software. The output port feeds the display device directly in serial form. The output data rate is controlled by the interrupt timer, and can be varied to suit the display used.

The complete schematic is shown in Fig. 1. Although it may look intimidating, most of the wiring is just parallel runs from chip to chip. Pins 26 through 33 of Z1 are the data bus over which instructions and data flow between the CPU and the other chips. The address of instructions, memory locations and I/O ports are output on the address bus, pins 9 through 20. The R/W line signals whether data is being read into or written out of the processor. Z5A and Z4B form a 1-bit output port which is connected to data bus 0. Z5B, Z6A and Q2 are used as an input port, driving data onto data bus 7. The 555, Z7, is an interrupt timer. Each time it clocks, the execution of the program is suspended and a new output bit appears at pin 5 of Z4. By varying the rate of the interrupts, the output rate may be set to any desired speed. More detailed information on the 6502 microprocessor is contained in the two excellent manuals available from the manufacturer (see Reference List).

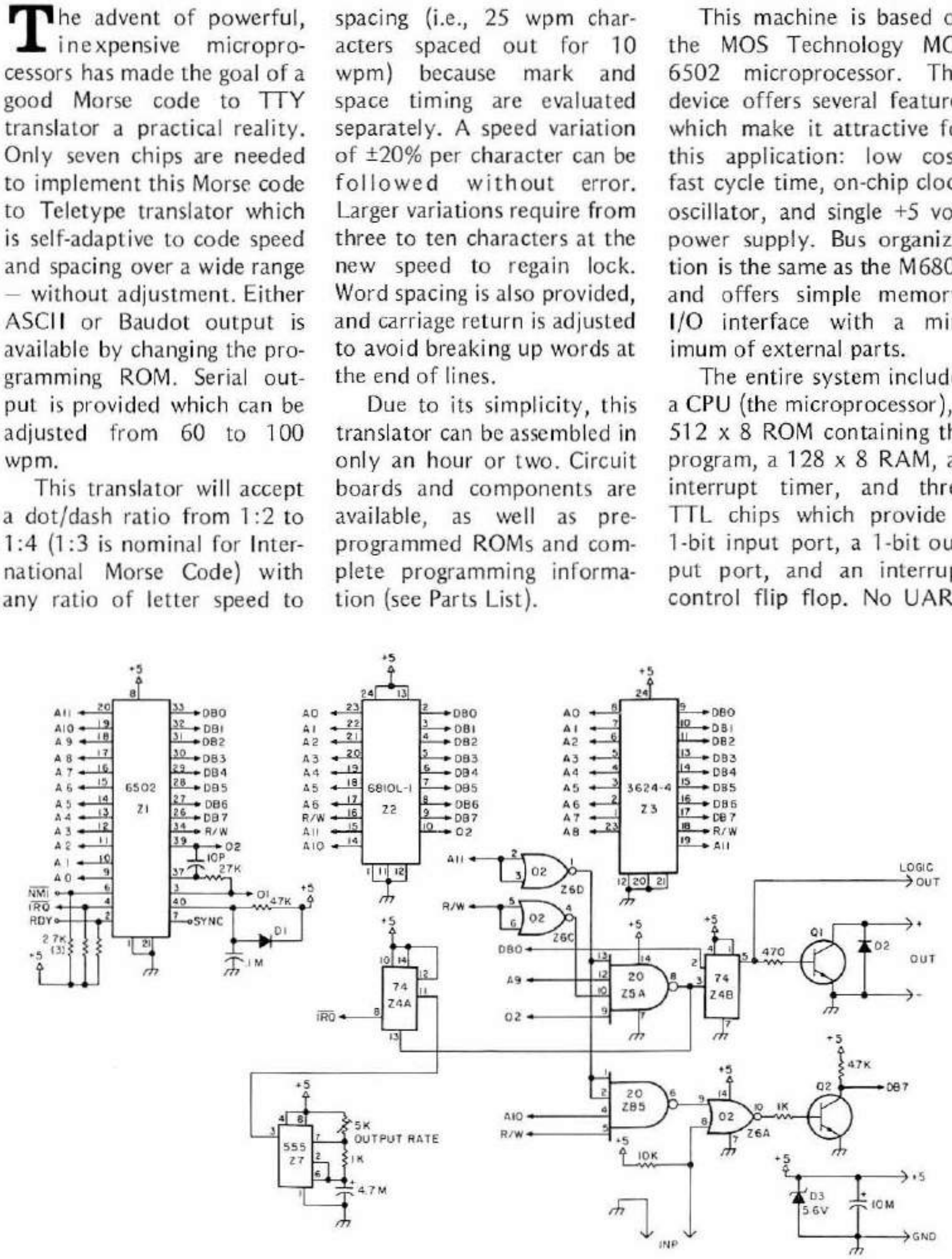


Fig. 1. Schematic.

The program is contained in the ROM, Z3. It is an Intel

D-3624, which is a fusible-link, bipolar, tri-state, 512 x 8 PROM. A complete program listing with notes is available, as well as pre-programmed PROMs. Basically, the dot-dash threshold is set equal to half the mark period if the element was a dash, or to twice the period if it was a dot. Thus, the threshold varies between 1/2 and 2/3 of dash time. This provides a tolerance in dot-dash timing, and makes the threshold follow the input code speed. A similar method is used for spaces. Word spacing threshold is initially set for 1-1/2 times the letter space. If more than eight characters are received without a word space, the threshold is decreased. If more than four characters are received with spaces between each, the threshold is increased. This process is iterated until proper spacing is achieved. A counter keeps track of how many characters have been printed in each line. At the first word space following the 64th character, a CR-LF-LTRS sequence is output. This can be easily modified for other printer or display widths, or bypassed entirely by minor changes in programming.

The input is sampled periodically by a software loop, which simultaneously increments a counter. When the input changes state, the elapsed time is represented by the number in the counter. The program processes the timing data, and then begins the timing loop over to wait for the next input change. The output is interrupt driven. Each time a new output bit is due, the timer Z7 clocks Z4A which in turn pulls the $\overline{IRQ}$ line of the CPU low. The microprocessor suspends whatever operation is being executed at the time, and jumps to a subroutine which outputs a new bit via Z4B. It then returns to the main program to resume operations. This procedure requires only 50 to 60 microseconds to complete, so there is no noticeable slowing of the main program execution. Part of the RAM is assigned as a FIFO buffer to accommodate situations such as upper case (FIGS-Char.-LTRS) or end of line (CR-LF-LTRS). In these cases several output characters are output very quickly, and the buffer stores and feeds them out sequentially at the TTY output rate.

Assembly

Construction is simplified by using the available circuit board, which is double sided with plated-through holes. However, wire wrap is also satisfactory. Lead dress is not particularly critical, except pins 39, 37 and 3 of Z1 and to the 10 pF capacitor and the 27k resistor. These are connections for the clock oscillator, and pickup can cause frequency modulation of the clock, which can cause some very strange problems. Reasonably short ground leads should be provided. Z3 in particular draws several hundred milliamps and should have appropriately heavy supply connections.

Parts List

Z1	MCS6502 (MOS Technology)
Z2	MC6810L-1 (Motorola)
Z3	D-3624-4 (Intel)
Z4	SN 74LS74 N
Z5	SN 74LS20 N
Z6	SN 74LS02 N
Z7	NE 555
Q1	2N5682 (Motorola, any 100 mA, 150 volt NPN silicon)
Q2	2N3904
D1	1N4148
D2	1N4004
D3	5.6 volt 1 Watt zener
1	470 Ohm 5% ¼ Watt composition resistor
2	1k Ohm
3	2.7k Ohm
1	10k Ohm
1	4.7k Ohm
1	27k Ohm
1	47k Ohm
1	10 pF ceramic NPO
1	4.7 mF 10 volt tantalum
1	10 mF 10 volt tantalum
2	.1 mF ceramic

Misc. sockets, terminals, circuit board

The following items are available from: Levy Associates
P.O. Box 961
Temple City CA 91780

Kit of all ICs, electronic parts, pre-programmed ROM, sockets, double sided plated through circuit board (4 x 6")	$139.00
Kit of all ICs including pre-programmed ROM	$ 99.00
Circuit board only	$ 17.50
Pre-programmed ROM only	$ 42.50
Complete program listing and notes (supplied free with any above)	$ 3.50

Special programming also available — write describing options needed.

Be sure to use sockets for all of the ICs. They must be inserted at various points in the checkout procedure. After mounting the sockets, they should be broken in by using the pins of a spare 14 or 16 pin IC. Insert and remove these pins at least once in each receptacle of the sockets, being sure to catch all of the pins of the 24 and 40 pin sockets. This reduces the insertion force required, and lessens the chance of bent pins and broken ICs.

A limited amount of parts substitution can be made. The 6810 must be a -1

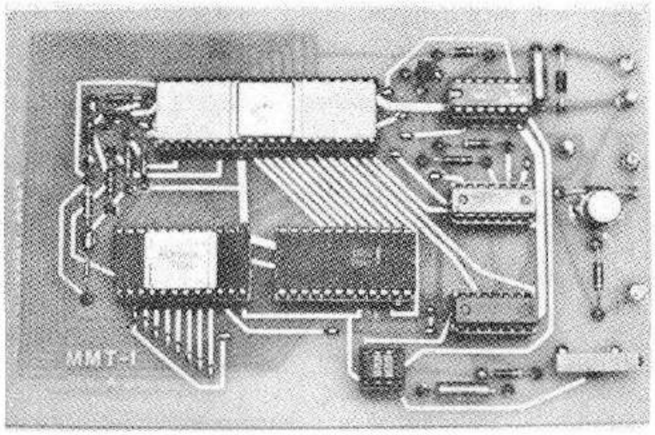


Fig. 2.

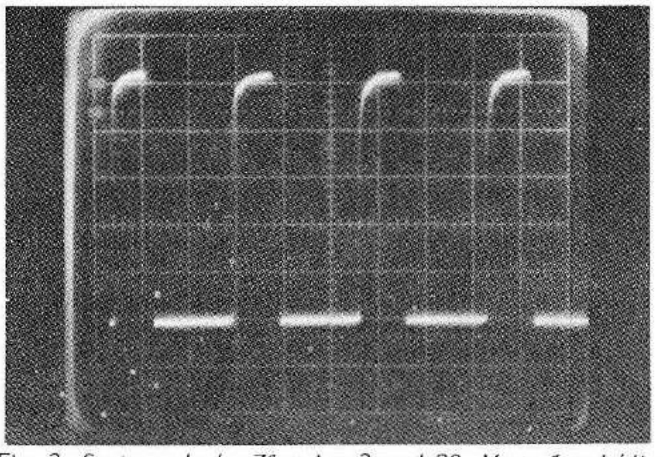


Fig. 3. System clock. Z1, pins 3 and 39. Vert: 1 volt/div. Horiz: 0.5 usec/div.

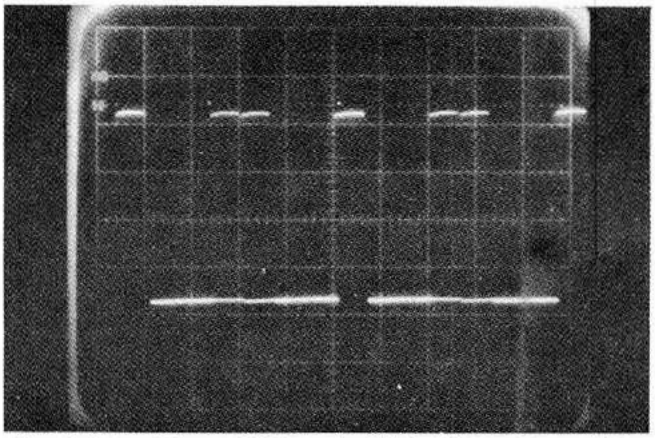


Fig. 4. Sync, Z1, pin 7. Vert: 1 volt/div. Horiz: 2 usec/div.

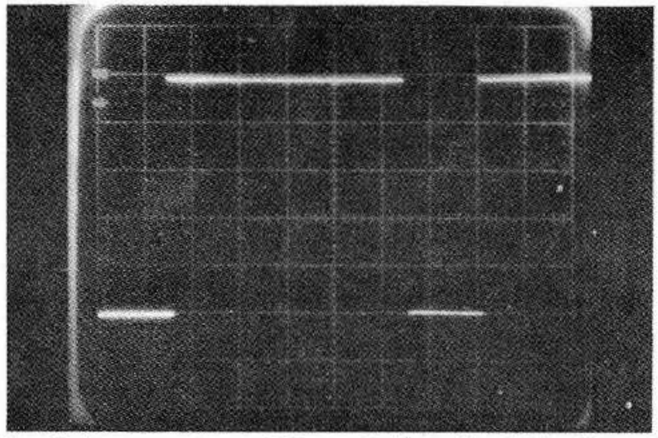


Fig. 5. Interrupt timer. Z7, pin 3. Vert: 1 volt/div. Horiz: 2 ms/div.

version, and the TTL must be low power Shottky to avoid excessive loading. The D-3624 can be either a standard or a -4 version, or a D-3604 can be substituted if eight 10k pull-up resistors are added to the data bus lines. Care should be taken to do the wiring with no mistakes. These components include relatively expensive MOS devices, which are not nearly so forgiving as TTL. Reversed polarity and shorted outputs can be disastrous. NOTE: Do not solder any part of the board without removing Z1 and Z2 first. Do not remove or insert any ICs while power is applied.

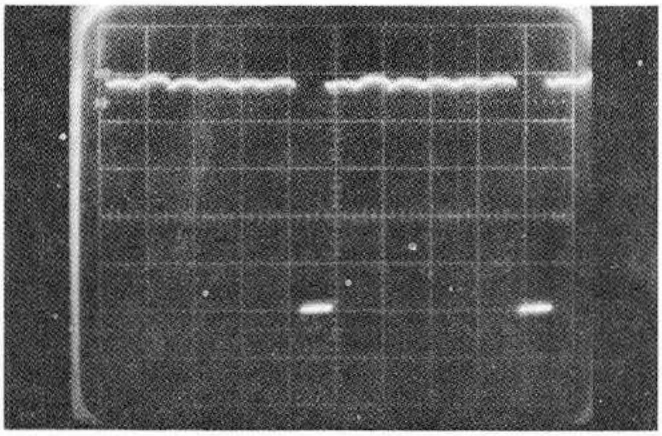


Fig. 6. Input sample pulse after 5 seconds. Z5, pin 6. Vert: 1 volt/div. Horiz: 2 usec/div.

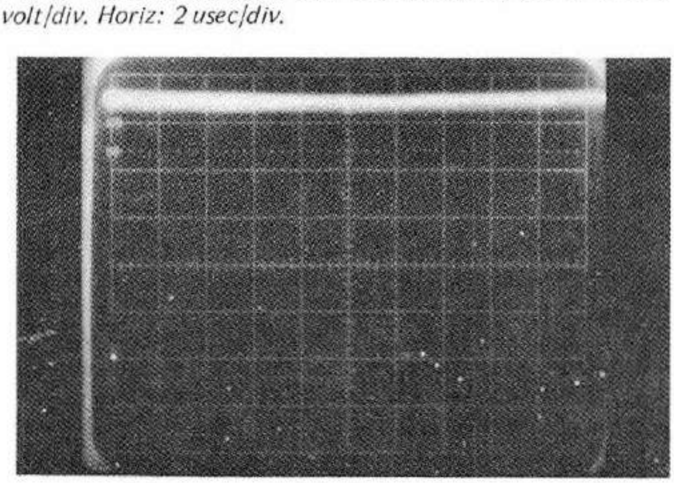


Fig. 7. Interrupt request, $\overline{IRQ}$, Z1, pin 4. Vert: 1 volt/div. Horiz: 2 ms/div.

Testing

A five volt, 400 mA current limited power supply and a 10 MHz scope, preferably dual trace, triggered sweep, are needed for testing. It should be noted that several of the ICs will get quite warm during operation. Z2 and Z3 will be almost too hot to touch after an hour of operation. The others will be slightly warm.

Remove all of the ICs and apply power. Check each socket to verify that +5 volts and ground appear on the correct pins.

Remove power and insert the 6502, Z1. Handle the 6502 by the pins in one hand and support the board on the other while inserting the chip. The conductivity of your body will keep static charges from damaging the MOS circuit. Be careful not to bend any of the pins. For all of the larger ICs it may help to carefully bend the leads on the two sides until they are parallel, to ease entry into the sockets.

Connect the scope to pin 3 of Z1, and apply power. An approximate square wave of about 750 kHz (±30%) should appear (Fig. 3). A similar waveform should appear on pin 39.

Move the scope to pin 7 of Z1. Using a clip lead, momentarily short the anode of D1 to ground. A burst of 1 to 2 microsecond pulses should appear (Fig. 4). The duration of the burst may be from a few milliseconds to continuous. Each pulse signals a new instruction being fetched by the microprocessor. Since there is no ROM to supply instructions, it executes essentially a random sequence, which usually "blows up" and stops executing after a period of time.

Next, insert Z2, the 6810-1, being sure to power off first. Use the same technique as for the 6502. Reapply power, and make the same test as before. The same results should be observed.

Insert the 555, Z7. On pin 3 of Z7 a pulse train will appear (Fig. 5). Set the trim pot for the time period corresponding to the desired output TTY speed from Fig. 10.

Insert the 74LS02, Z6, the 74LS20, Z5, and the D-3624 ROM, Z3. Connect the scope to pin 7 of Z1. You should see a series of 1 microsecond pulses of irregular spacing (Fig. 4). Connect the scope to pin 6 of Z5. Momentarily short the anode of D1 to ground. Immediately following, a 1 microsecond pulse with a repetition rate of approximately 7 milliseconds should appear. This is the input sample rate. After a few seconds, it will change to a repetition rate of 8 microseconds (Fig. 6). Each time the system is reset, either by shorting D1 or by reapplying power, this pulse should repeat the same slow, then fast, repetition rate sequence. At this point the CPU is fetching and executing instructions properly.

Speed (wpm)	Z7 Period
60	22.22 ms
66	20.20 ms
75	17.78 ms
100	13.33 ms

Fig. 10. Z7 timing period for TTY output speeds. One new bit is output for each pulse for Z7. Higher speeds can be obtained by changing the timing components.

Insert Z4, the 74LS74. Connect the scope to Z1, pin 4. A negative-going pulse of approximately 50 microseconds duration should appear, with a repetition rate equal to the period set by the trim pot earlier (Fig. 7). Careful examination of this pulse will show some jitter in its width (Fig. 8). This amount should be less than 1% of the pulse to pulse spacing.

Connect a key from the input to ground. (If a keyer is used, be sure it is set up to switch positive voltages to ground.) Put the scope on the logic output pin. Set the timebase to 20 milliseconds/division. Send CQspaceCQspaceCQspace . . . After a few characters, the beginning of each character should trigger a series of pulses (Fig. 9), with the series following the space being twice as long (because it is Character-Space sequentially). These series of pulses are the serial TTY output.

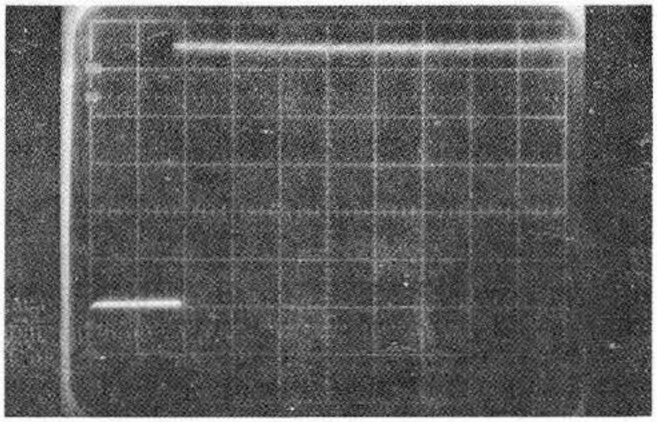


Fig. 8. Interrupt request, $\overline{IRQ}$, Z1, pin 4. Vert: 1 volt/div. Horiz: 20 usec/div.

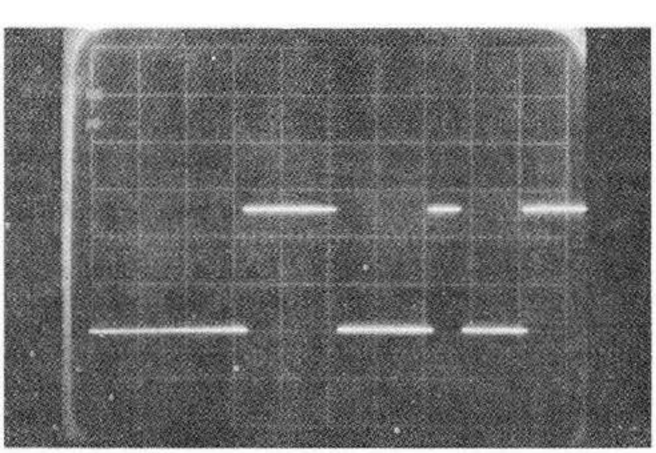


Fig. 9. Logic output. Vert: 1 volt/div. Horiz: 20 ms/div. This is the serial TTY output (set for 100 wpm).

Connect the TTY machine or display as shown in Fig. 11. Be sure to watch polarity and grounds. Fig. 11 also shows the configuration for TTL compatible outputs. Now the machine should print the correct translation of the input code. Word spacing will take three or four words to settle down. An automatic CR-LF-LTRS sequence will be output after the first word space following the 64th character of each line. Unless an unusually long word (such as microprocessor) is sent, this will prevent breaking up of words.

The printer will normally be one character behind. This is because it is still measuring the space until the next mark comes along. Each character will be output as soon as the following one starts.

There is a built-in buffer of approximately 40 characters which is designed to accommodate extra characters like FIGS, LTRS, CR, LF, etc. It will also allow input code speed to exceed TTY output speed for a brief time. The speed range is limited on the high end by the TTY machine speed used. For a 60 wpm printer, it will limit at about 50 wpm code speed, and for a 100 wpm printer, about 65 wpm code speed. A programming change to increase the sample rate used at the input will allow a higher upper limit, with a corresponding increase of the minimum from the normal 5 wpm.

There are two things to check in case of difficulty. First, be sure you are not running your letters together. It is a natural tendency to try to send as fast as possible to test this type of system, and this usually results in running letters together. Remember, the machine is relatively dumb, and doesn't know that – . . . . is really "the" and not "6" because you don't say "6 real thing."

The delayed audio feedback caused by the printing delay can also raise havoc with your timing. A pair of headphones or a silent printer is the cure.

The other thing to check is that the output speed is set carefully to match the TTY machine speed (within 1%). A clue to mismatch here is consistent errors such as extra Ts after, or instead of, spaces, or CR instead of spaces, etc. Also check to be sure the loop supply being used is compatible with the TTY machine. Newer models are fairly tolerant of open circuit voltage and exact current, but older machines are not. Substituting a keyboard for the translator is a quick way to check this.

Conclusion

Your Morse code translator will now operate from a key or keyer. I will be happy to attempt to assist in case of difficulty, but please send an SASE. ■

Reference List

1. MCS 6500 MICROCOMPUTER FAMILY PROGRAMMING MANUAL, August 1975, MOS Technology Inc., Norristown PA.
2. MCS 6500 MICROCOMPUTER FAMILY HARDWARE MANUAL, August 1975, MOS Technology Inc., Norristown PA.

Both of the above manuals are available for $5.00 each from MOS Technology Inc., 950 Rittenhouse Road, Norristown PA 19401.

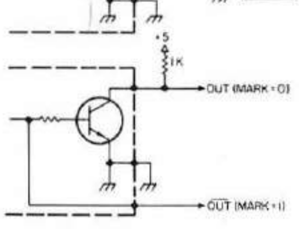
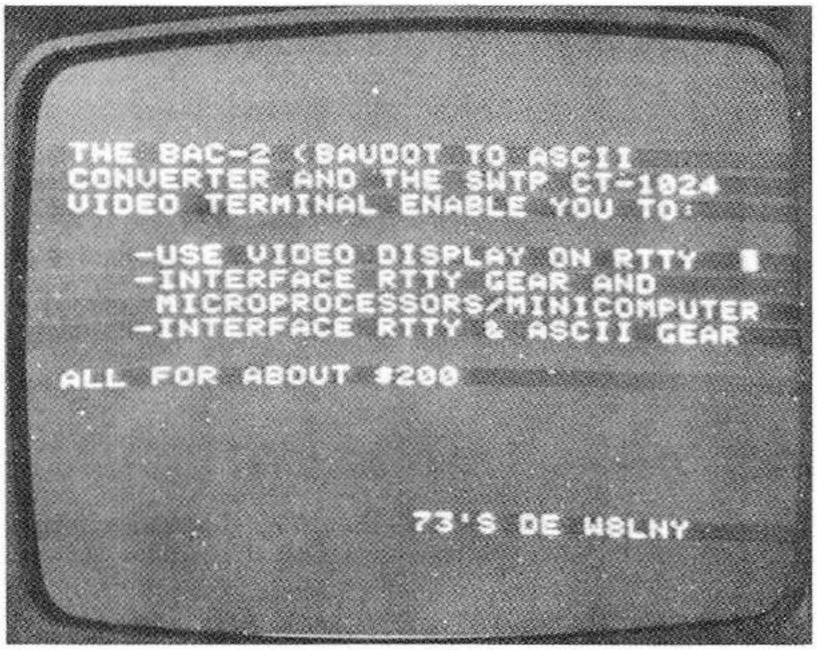


Fig. 11. Output connections.



THE BAC-2 (BAUDOT TO ASCII
CONVERTER AND THE SWTP CT-1024
VIDEO TERMINAL ENABLE YOU TO:

-USE VIDEO DISPLAY ON RTTY
-INTERFACE RTTY GEAR AND
MICROPROCESSORS/MINICOMPUTER
-INTERFACE RTTY & ASCII GEAR

ALL FOR ABOUT $200

73'S DE W8LNY

R. David Guthrie W8LNY
2956 London Wall
Bloomfield Hills MI 48013

ASCII/Baudot with a PROM

-- for ribbonless RTTY or computers

Video displays are becoming increasingly popular in RTTY. In addition to eliminating the noise of mechanical printers (appreciated by the XYL), there is no mechanical maintenance or paper/ribbon changing. The video user has the ability to rapidly change baud rates for use in copying commercial stations and to use the video unit together with microprocessors.

Commercial kits start at $395 and up. This discourages many RTTY enthusiasts. Fortunately, Southwest Technical Products has a video display unit for $190.50 with power supply called the CT-1024[1]. It requires the user to accomplish his own Baudot to ASCII code conversion, which is the subject of this article. The result will be very close to the commercial units available with 3 exceptions:

1. *Page Size.*

HAL RVD-1005 – Has 40 characters/line, 25 lines for a total of 1,000 characters ($575); Leland – Also has 40 characters/line, 25 lines ($395 in kit form); CT-1024 – Has 32 characters/line, 16 lines for a total of 512 characters. Another 512 are stored for a 2 page memory.

2. *Scrolling.*

Both the Leland and HAL units scroll up the page. That is, each new line appears on the bottom and pushes the remaining lines up one. The SWTP CT-1024, on the other hand, wraps continuously around the screen. When it gets to the bottom, it starts over again at the top of the screen. The blinking cursor identifies the current position.

3. *Treatment of words at end of line.*

Both the HAL and Leland units start a new line somewhat prior to the actual end of the line on a space character, so as to not separate a word in the middle. This is not available on the CR-1024. It would be relatively easy to add with some counter ICs if needed.

Basic Display Overview

Fig. 1 shows the functional flow of data from the RTTY terminal unit to the video display. At this point, you have the equivalent capability of the commercial displays previously discussed.

Fig. 2 shows three ways of getting from your present RTTY setup to the TTL logic level signals required by the BAC-2. Note that the output of the ST-5/ST-6 terminal units is the same as the RS-232C commercial data standard so that modem out-

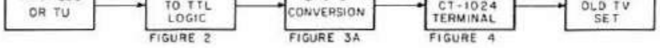


Fig. 1. Overview – basic RTTY use only.

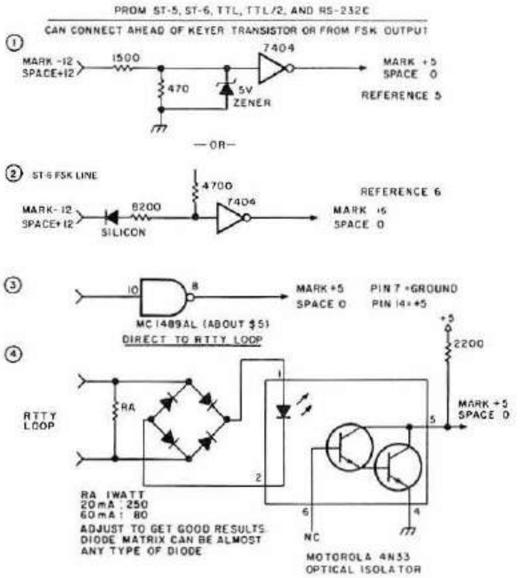


Fig. 2. Connection from RTTY loop to BAC-2.

puts could be treated the same way. Two of the circuits are described further in the references. The current loop interface can be used with any terminal unit and may be of interest to non-RTTY microprocessor users who want to use Baudot teletype equipment. The current loop interface is totally isolated from the loop.

Fig. 3A details the heart of the unit – the Baudot to ASCII converter – dubbed the BAC-2 (someone has probably designated a BAC-1!). The circuit provides:

1. Serial to parallel conversion of incoming Baudot signals using a UART and multiple speed clock for various baud rates.
2. Conversion to ASCII using standard 8223 PROMs (programmable read only memories).
3. Letters/figures case shifting from the 8223s to U4 wired as a cross coupled flip flop.

Fig.,4 details the connection of the BAC-2 basic unit to the SWTP CT-1024 terminal unit. Keyboard connections are specified – the user can also go in via the pins on the back of the unit. With parallel data, keep the lines between the BAC-2 and CT-1024 *extremely* short. Over 18 inches and ringing is likely to occur. The BAC-2 might be mounted right on the CT-1024 as shown in the picture.

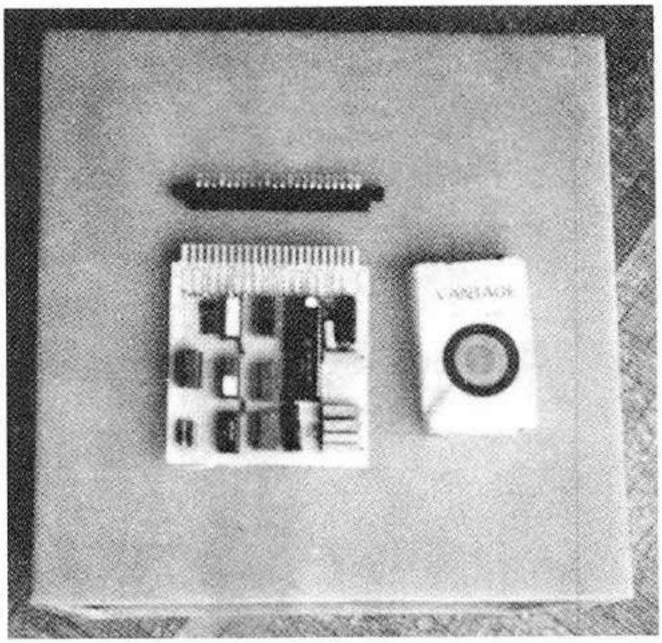
Optional Circuits

Up to this point I have described using the BAC-2 primarily for RTTY copying. This will undoubtedly serve the needs of many readers. However, with microprocessors becoming more available and inexpensive every day, you might want to plan ahead now. I have a microprocessor using the Motorola 6800 CPU that provides all of the functions of RTTY Selcal, CW and RTTY ID, FIFO/UT-4 buffer storage, automatic transmitter control, ASCII-Baudot code conversion, and numerous other RTTY goodies. Use of a microprocessor is very easy and far less complex than wiring up a lot of discrete components for functions. Moreover, the functions can be changed much more quickly and easily by just changing the program.

If your future plans call for a microprocessor, you should plan now to get into serial mode rather than parallel mode for transferring data. Parallel mode has two disadvantages from my point of view:

1. Line lengths and interfacing techniques get tricky as you add devices and circuits. Serial start-stop, on the other hand; is very forgiving and insensitive to glitches on the leading and trailing edges of pulses.
2. It is a snap to switch electronically between several input and output devices in serial mode. With parallel data either data selectors (such as the 74157) or tri-state buffers are needed. Further, the timing sequences for strobe pulses to indicate a character ready get pretty tricky with tri-state devices. Timing chips and extra circuitry will quickly cost far more in dollars and time than a couple of UARTs.

Fig. 5 shows a functional overview using the BAC-2 in a RTTY system with serial data flows. All data is flowing in TTL logic format (Mark = +5 V, Space = 0 V). Other units you may have can be readily added by converting them from/to serial mode. While the NAND and inverter gates don't really belong in a functional diagram, they help show how the pieces are integrated. The only things needed to be added for complete versatility are some switches to select/control the serial data flows as needed for use with the microprocessor. They can be omitted at this point for display use only. Note that the 8th or parity bit is ignored in all ASCII circuits. The steps through Fig. 3A are the same as the basic unit discussed previously. In Fig. 3B a PISO (Parallel to Serial Output) circuit is shown. The NE555 timer should be set to 110 Hz for 110 baud or 300 Hz for 300 baud use. It is simple and reliable and is discussed in depth in the *TTL Cookbook*[2]. Note some deviations from the *TTL Cookbook* diagram – the circuit as shown in the book does not work. Probably some minor typos in printing of the book. The same circuit is shown again in the path from the ASCII keyboard (if used) to the serial data flow. Use of an optional circuit between the keyboard and PISO to do things like repeat key, counting characters, and so forth may be desirable and is discussed in a past issue of *73*[3].



The BAC-2 and PISO Optional Circuit Board. A pack of cigarettes is included to show size and is not required for successful operation.

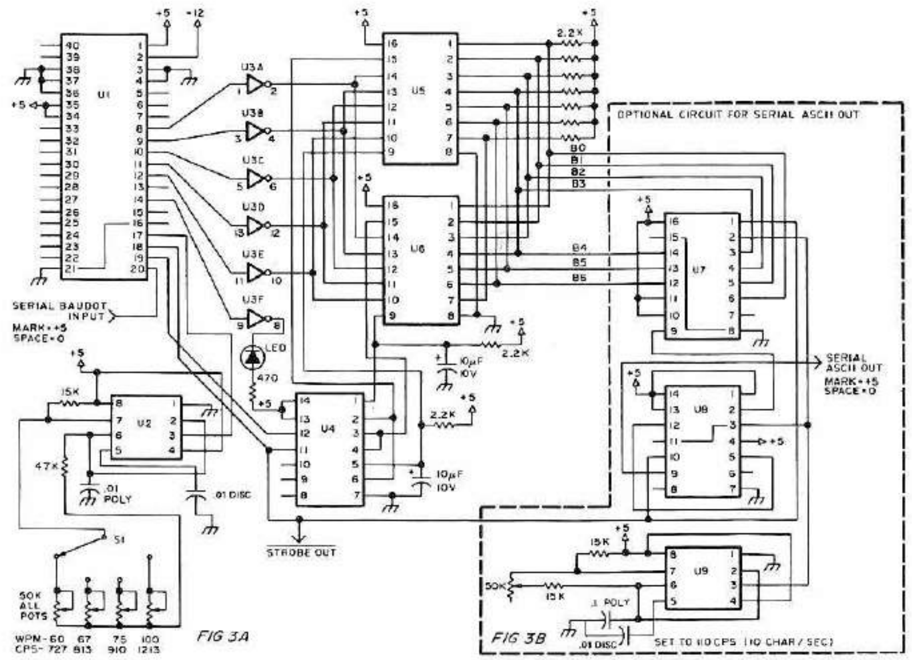


Fig. 3. Baudot to ASCII converter. U1: UART (see text). U2: NE555 timer. U3: 7404 6-inverter. U4: 7400 4-NAND. U5, U6: 8223 PROM. U7: 74165 PISO register. U8: 7474 flip flop. U9: NE555 timer. All resistors: ¼ W 20%. S1: single pole, 4 position rotary. LED: any LED.

Use of two gates provides the mixing of ASCII signals from the keyboard, BAC-2, and/or microprocessor. If not using a micro at this time, simply leave the connections open and tie the ASCII from the microprocessor to +5 V for now.

Fig. 6 shows a serial to parallel (SIPO) converter to front end the CT-1024 terminal. Not much to hooking it up. SWTP also has a CT-S kit available for $39.95. It involves a lot of conversions from/to the commercial RS-232C standard, which is nice if you use it. The SIPO circuit can be built for less than $10 depending on where you get the UART.

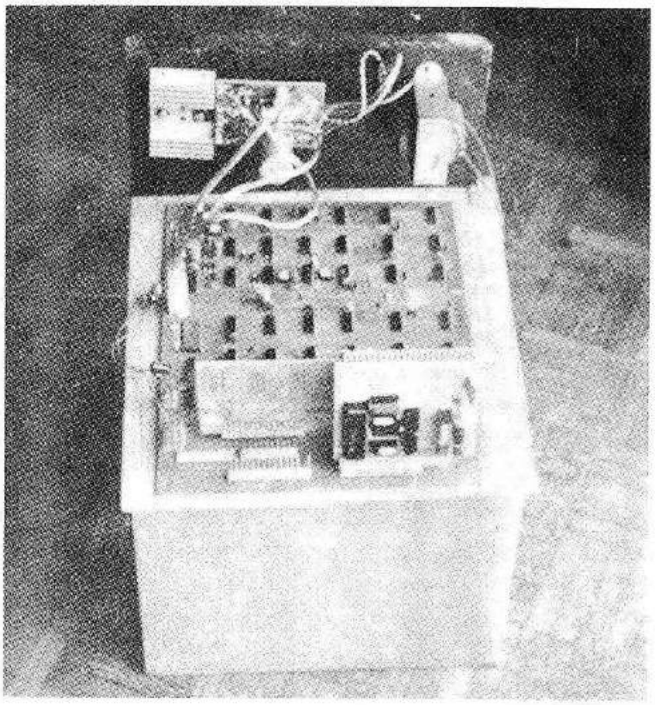
We use Fig. 4 again to interface from the SIPO to the CT-1024 terminal. Same leads and terminology — just a different source.

Fig. 7 is optional and converts the serial data from TTL to the commercial RS-232 standard for use with an "outside world" terminal such as commercial computer video displays, printers, or 33 teletypes with RS-232 conversion built into the TTY.

At this point, you are probably wondering where the ASCII to Baudot circuit is. We have everything flowing everywhere but back to the teletype. The answer is there isn't any! The microprocessor provides the tool for the code conversion and the FIFO type buffering of data. If you don't plan on a micro, a circuit to convert from ASCII to Baudot can be constructed and was described in a recent *73* issue[4]. The micro also provides for CW/RTTY ID, "Here is" Key, and other goodies to the RTTY system.

Construction of the BAC-2

The circuit was constructed using wire wrap methods. Not only is it inexpensive, relatively fun (you can overlap wrapping and watching TV at the same time with the family), but also totally flexible as far as experimenting and making changes. Requirements are a wire wrap tool[7], some wire (#30 Kynar — $10 for 1,000 feet usually, or available prestripped), a wire stripper (filing a notch in a pair of needle nose pliers with cutter is good), wire wrap sockets, and a board. Radio Shack has



The CT-1024 Terminal Unit mounted on a board with power supply in rear. A SIPO converter is standing upright on the back of the unit.

BAC-2/SIPO	CT-1024 Connector J9
B0	1
B1	4
B2	5
B3	7
B4	8
B5	11
B6	12
$\overline{\text{STROBE}}$	10
+5	2
−12	6
GROUND	3

CT-1024 Connector J3

Jumper pin 3 to pin 7 to force erasing each line prior to data entry.

CT-1024 Wiring Option

Jumper 1 to 3 for negative going strobe.

Fig. 4. BAC-2 parallel interface to SWTP CT-1024 terminal.

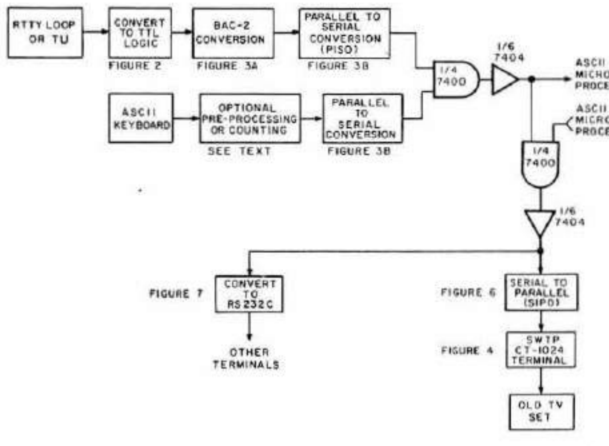
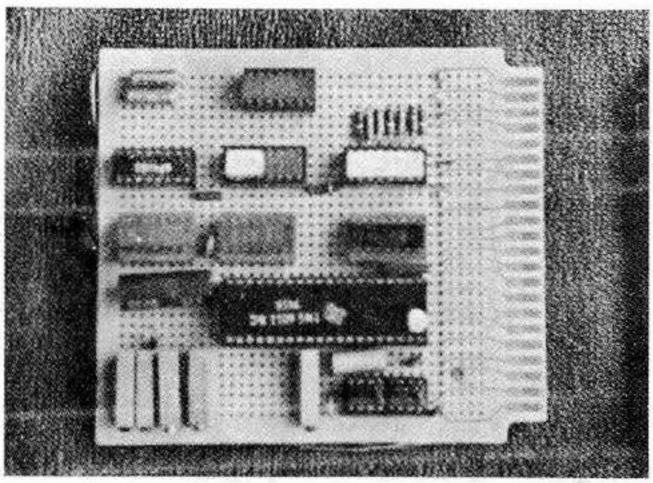


Fig. 5. Functional overview – serial option.



The BAC-2 converter board (subject of this article). The one chip on the side opposite the 5 trim pots is not part of the article circuit (corrects an error in programming my PROMs).

a new board (276-152) for $2.99 which provides space for IC sockets and a dual 22 pin edge connector. The mating connector (276-1551) is also $2.99. Watch out for some boards that have holes every .200 inches rather than .100 inches per the catalog specs. Somehow these crept into stock in a few stores and should be avoided. The board includes the PISO circuit of Fig. 3B. You may want to consider crystal control of the baud rates using circuits that have appeared from time to time in the *RTTY Journal* or *73 Magazine*, if a counter is not available. I have had no trouble with the NE555 timers. Of interest in Fig. 3A is an LED indicator connected to pin 14 of the UART. If it flashes, the incoming data has no valid stop bit. This is super for copying commercial stations with varying shifts, baud rates, and upright/upside down shifts. Just experiment with the controls until the bad stop LED goes out.

Readers who have built a UT-2/UT-4, or other equipment incorporating a UART, can skip U-1 and U-2 in the diagram and connect directly to the code conversion circuits in parallel form. Likewise, the UART can be used to up speed convert at no extra charge by wiring up the transmit side of the UART to convert all Baudot to, say 100 wpm.[6] The transmit side of the UART cannot be used for the serial ASCII output in Fig. 3B due to the limitation that it be used with either 5 or 8 bit code on both sides (you can't split it). When a valid character has been received by the UART, the data received line (pin 19) goes high. It is inverted and connected to the data received reset (pin 18) to reset the UART. The data is available prior to the data received line going high and stays so until the next character comes along allowing plenty of time for the 8223 PROMs to convert the code. The reset (pin 18) gives us a strobe line that is negative going to trigger the PISO circuit or the CT-1024 terminal depending on hookup.

The received data (pins 12-8) of the UART are buffered by 7404 hex inverters to protect the UART, provide the drive for more than one code converter chip, and allow use elsewhere.

Code conversion is accomplished by two 8223 PROMs. These are available from a variety of sources. A self-programming technique was described as a part of another circuit in *73*.[8] The author of that article also mentions references for programming. Fig. 8 shows the bits to be programmed (blown to 1) on the 8223s. These are the Mark-Sense cards used by Hamilton-Avnet, a major wholesale distributor of ICs. Hamilton charges $8.95 each to provide an 82S23 (newer version of the 8223) already programmed ($6.45 + $2.50 programming) per the specs of Fig. 8 if you have prepared the card. The cards shown are for unshift on space – bit 7 is used to select the other chip when going from letters to figures. Terminology for the two chips is described in Fig. 8 for communication purposes together with the distinction between having unshift on space or not. If you want the option of switching between unshift on space or not, you might add another 8223 with a switch to the enable line (pin 15), or add some gates to select the

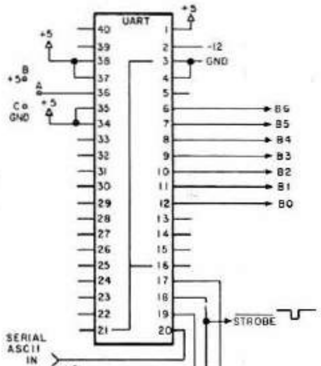


Fig. 6. Serial in/parallel out (SIPO) for ASCII code. Jumpers: A – B, 2 stop bits; A – C, 1 stop bit. Adjust for 16x baud rate: 110 baud = 1760 Hz; 300 baud = 4800 Hz.

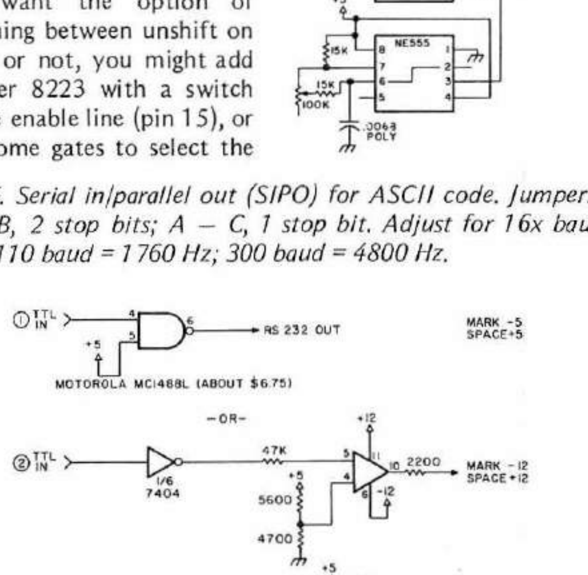
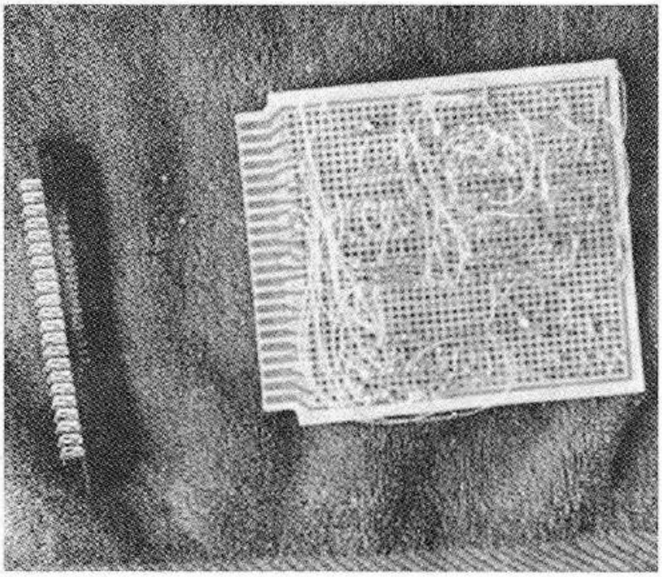


Fig. 7. TTL to RS232-C conversion.



Flip side of BAC-2 board. The wire for wire wrapping is a very pale yellow and doesn't show very well.

character yourself. Note that the Baudot control functions, Letters, Figs and Blank, are all translated to the ASCII Rubout (all 1s) character to be ignored by either your microprocessor or video display. The two 10 uF capacitors prevent the flip flop for letters/figures from triggering as the 8223s change state.

The CT-1024 Display Unit Construction

The SWTP display unit is available in kit form for $190.50.[1] Boards only are also available for $47.50. The ICs and discrete components can be purchased from *73* advertisers for $50-70 in total. Readers with a well-endowed junk box might want to go the board route to save money. Caution: The boards are very complex and built to excellent and exacting standards by SWTP. I would not recommend attempting to make them yourself or to purchase them from anyone other than SWTP. *Radio Electronics*[9] magazine carried a series of articles on the CT-1024 if you want to read more prior to making a decision (the TV Typewriter II in the series). I don't recommend the TV Typewriter I (Sept. 73 *RE*), as some additional interfacing is needed for it to properly handle the line feed, carriage return, no display of rubouts, scrolling, clear, etc. The TVT I is tough to use either with a micro or RTTY without extensive rework.

A feature not described by SWTP is to tie pin 3 to 7 of the cursor socket (J3). This causes the CT-1024 to clear the next line upon receipt of a CR/LF combination. Without this, it provides a very confusing display. It automatically starts a new line and clears it when the end of a prior line is reached (non-overline) when wired this way.

Assuming that you have built up the CT-1024 in one form or another, the next step is to interface it with a TV set. The CT-1024 instructions provide an interface to a Motorola TV. Lancaster[10] discusses interfacing video signals to TV sets in depth in a recent BYTE article, and will have this available in a "TV Typewriter Cookbook" shortly. You should be able to test the CT-1024 by grounding the data line that you want to be a zero and then grounding the strobe line to see that it is working. An ASCII keyboard could also be attached to check it out.

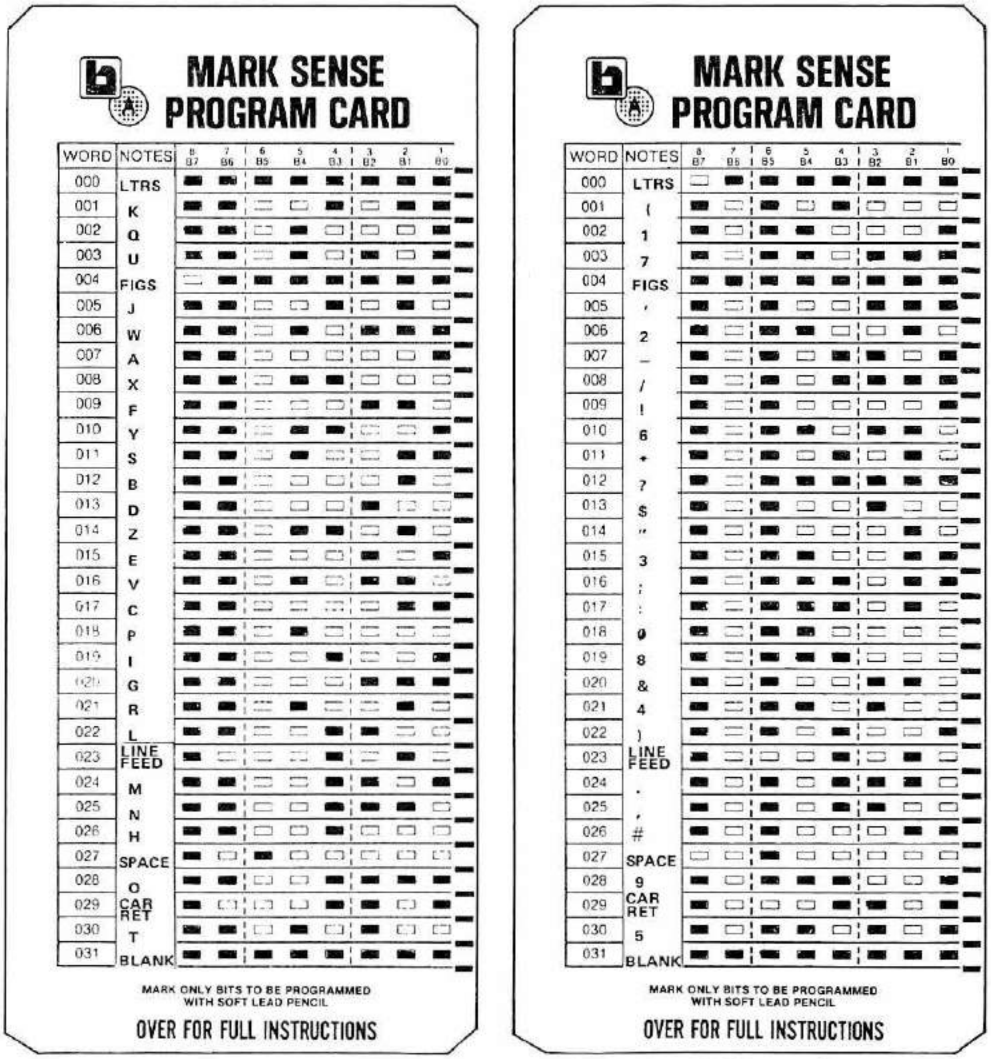


*Fig. 8. 8223 read only memory coding. Bell prints as *. Blacken B7 (first box) to block unshift or space.*

The SIPO (Serial In/Parallel Out) Circuit

The circuit in Fig. 6 can be readily constructed on perfboard or the same Radio Shack boards discussed previously. Set the NE555 timer rate to 16x the baud rate; i.e., for 110 baud/10 characters per second, 1760 Hz would apply. Jumper A to B for 2 stop bits (110 baud), and A to C (300 baud +) for 1 stop bit as necessary.

Conversion to RS-232C Standard Data Logic

Fig. 7 shows possible ways to convert from TTL to RS-232C logic. Either results in pretty much the same thing. The trade-off is what power you have available versus using discrete components.

UART Considerations

The UARTs specified in the various circuits are the "standard" variety. Makers and models are as follows:

1. Texas Instruments – TMS6011.
2. General Instrument – AY-5-1013 or AY-5-1014. The 1014 is nice in that no –12 V is needed – pin 2 is left unconnected.
3. Western Digital – TR1602B.

These are readily available from *73* advertisers and other sources.

Summary

I have discussed various ways of setting up the conversion from Baudot to ASCII and the video display. Hopefully, you will be able to select the options that you want. The Baudot to ASCII circuit (BAC-2) in and of itself may be useful for working with a microprocessor in that the program to convert back again can be readily loaded from 5 level paper tape using Baudot equipment. Once loaded, the conversion back can be done via the program. This enables SWTP M6800 microcomputer system users to use 5 level equipment and avoid going to ASCII equipment. I do not have a source for printed circuit boards – let *73* advertisers know if you need them and perhaps someone will make them up. ■

References

1. CT-1024 Terminal, Southwest Technical Products, Inc, 219 W. Rhapsody, San Antonio TX 78216. Prices for:

A. Complete Kit	
CT-1024 Terminal System Kit	$ 175.00
CT-P Power Supply Kit	15.50
	$ 190.50
B. Boards Only	
CT-1024 Board Set	$ 47.50
MP6800 Processor Connector Set (provides 50 pins and mating molex connectors)	2.50
	$ 50.00

2. *TTL Cookbook,* Don Lancaster, Howard W. Sams & Co., Inc., 1974, pp. 262-265.
3. "Using a Bargain Surplus Keyboard," Cole Ellsworth, *73 Magazine,* January, 1976, p. 212.
4. "ASCII to Baudot Converter," Cole Ellsworth, *73 Magazine,* February, 1976, p. 52.
5. "RTTY Autocall – the Digital Way," L. W. Sanders, *73 Magazine*, February, 1976, p. 76.
6. "The Mainline UT-4," Irvin M. Hoff, *RTTY Journal*, March, 1975, p. 4.
7. Hand Wire Wrap Tool, Cambion #601-2506. Less than $3 from distributors.
8. "The Computer QSO Machine," B. D. Lichtenwalner, *73 Magazine,* January, 1976, p. 80.
9. "TV Typewriter II," *Radio Electronics*, Ed Colle, February/March/April, 1975.
10. "Television Interface," Don Lancaster, BYTE Magazine, October, 1975, p. 20.

I/O Editorial
from page 77

new industry started . . . the hardware and software background combined. In five years the big manufacturers in the micro field may well be the small firms which are springing up today to provide hobbyists with equipment.

It is possible right now to develop a computer product and get into production with it for very little . . . and do well. Never has there been such an opportunity in the computer field for small businesses to start and grow. An investment of a couple of thousand dollars today could well launch a company worth over a million dollars in just a year or two.

Well, that's the message I thought the computer hobbyists and computer store owners might be interested to hear. Time will prove whether I am overly optimistic or not. My past predictions have come off well, so maybe I'll hit again.

PROGRAMS FOR SALE?

Newcomers to the computer field are sometimes surprised, once they have their system up and running, to find that all they have is machinery and the darned thing just sits there . . . not *doing* anything. Not that everyone hasn't heard of software and programs – it's just that many neophytes don't understand the tremendous importance of such.

Old hands are culpable in this, too, for they are generally familiar with a commercial or school system which came with software and, shucks, you can get all sorts of programs from the users groups . . . from libraries . . . no strain. Oh yeah?

Foreseeing this problem last year, I organized the Kansas City meeting of the microprocessor industry to form a standard cassette medium for program interchange. The meeting was difficult to get going . . . I had to get the site set up for it . . . send out letters to all involved . . . write it up in *Byte* . . . and then make an awful lot of phone calls to follow it up. The meeting, for which *Byte* has been given credit, was my idea and my doing . . . and the people who had taken over *Byte* were furious that I had set up the conference . . . they didn't want to be bothered.

So much for that.

So now we have a medium which can be used for entering programs into small computer systems. One thing which I think will help hobby computers and small business computer systems grow more than anything else will be a low cost plentiful supply of programs. I hope to provide same.

Here is the deal. I will buy programs from programmers and pay a good royalty on them. I will duplicate these and have them on sale in all of the computer stores around the country . . . and possibly in some of the more forward looking electronics distributors. I will have them carefully checked in my own lab to make sure that they work as advertised and that duplicates are perfect. We already have excellent tape duplicating equipment which we use for making the Morse code and radio theory cassettes we are presently selling . . . over 2500 a month.

If you are a programmer and have a good program available, please let me know. I need to know what it can do, what system it is for, and how much memory it takes. We will be able to check out programs on the Altair 8800 . . . have been promised a Sphere system . . . and hope to eventually have all other popular systems set up and running so we can check the programs.

What kind of programs can sell? Just about anything. I would expect we might have a cassette with five games on it which would go for $2.95. A real fine Star Trek program might go for $4.95. A payroll program for small businesses could go for perhaps $19.95. We would want to keep the prices down to where it would not be worthwhile to bootleg them or even make copies for friends . . . thus getting out from under the problem MITS has had with their BASIC, which apparently has been copied quite a bit.

Programmers would benefit substantially under this plan, since they would make 10% of the gross sales. Thus a $12 program would wholesale to computer stores for about $8 and the programmer would make an 80¢ royalty. This would mean $8,000 royalty for every 10,000 of the programs sold . . . and with several hundred thousand customers out there, if only 10% of them bought a given program this would mean a very nice return for the effort of writing and documenting the program.

These programs would be a bonanza for the stores . . . it would give them the ammunition they need to sell systems . . . not to mention that they might eventually be a good profit item in themselves as businesses and hobbyists come into the store to find out what new programs are available every week or so.

It is going to take this idea a while to get into motion. First we will have to get our own computer systems up and working . . . our experience has shown this not to be a minor undertaking. We have two computer techs and programmers on our staff right now and are looking for more. Then we will have to run thorough tests on cassettes to see which work best . . . using a computer to check each tape as it is made, comparing it, bit for bit, against the original.

Programmers . . . it's time to get working on your programs. Be sure to document them well . . . be sure you are not drawing on others' work . . . all our programs will be copyright . . please put in some extra steps

Continued on page 93

ASCII to Baudot Converter

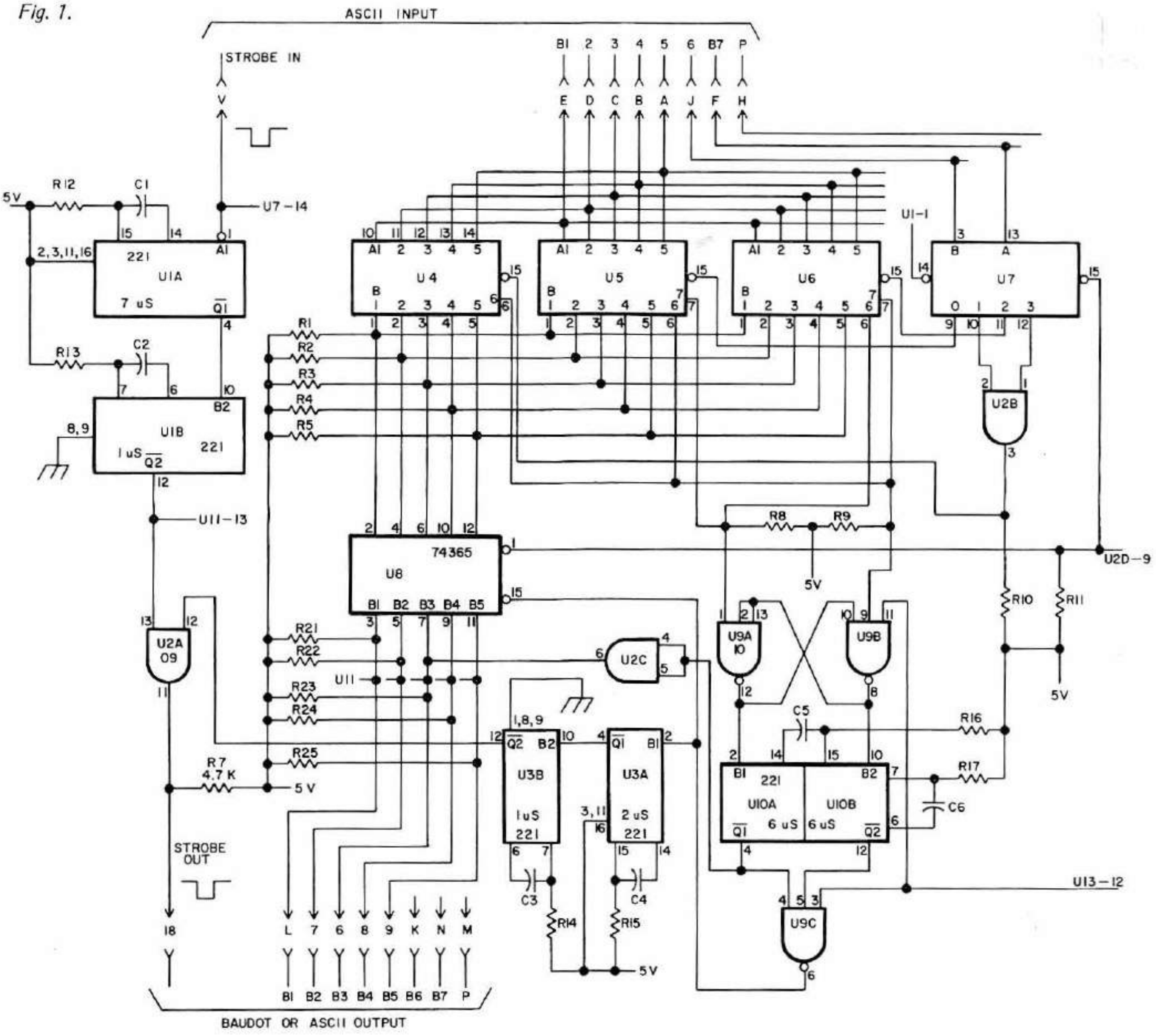
by
Cole Ellsworth W6OXP
10461 Dewey Drive
Garden Grove CA 92640

After acquiring a surplus ASCII encoded keyboard[1] it was desired to use this keyboard on the amateur RTTY frequencies. Under present FCC regulations, only 5-level code is permitted (commonly called Baudot code, but actually a version of the Murray code). Therefore, we needed a conversion device that would change the 8-level ASCII code to 5-level code. There are several different approaches to this problem presently in commercial use. One of the earlier conversion methods utilized tape reperforators and tape readers to accomplish the conversion. The more modern video display communications terminals in some instances use computer memory and software to make the conversion. Some integrated circuit manufacturers have made available commercial versions of custom programmed ROMs which greatly facilitate a bi-directional conversion but seem to be somewhat different in format from my requirements, including low cost.

The ASCII code is capable of generating 128 characters (2^7), of which up to 96 may be printing characters. (The remaining 32 are termed "Control" characters.) The Baudot code is capable of generating only 32 characters (2^5) so a "Case Shift" method is used to provide a second set of 32 characters while maintaining a 5-bit code. Thus the need for FIGS shift and LTRS shift on 5-level machines is apparent. It is the requirement for generation of the case shift character that makes the problem of conversion from ASCII to Baudot so interesting. Conversion in the opposite direction, i.e., from Baudot to ASCII, is relatively simple. Witness the recent publication of a circuit[2] that requires only four ICs to perform the conversion.

I had been mulling around several possible approaches to the conversion problem when two significant situations arose which served to solidify the design approach. The first was during the course of a discussion of the matter with Jerry WB6WPX, when he suggested "jamming" the case shift character into the Baudot output bit stream just ahead of the character requiring the case shift. The second was the development of the UT-4 by Irv W6FFC[3]. The FIFO in the UT-4 makes the perfect buffer for absorption of the case shift character (which is generated within the period of a few microseconds) without

Fig. 1.



significant delay in conversion of the following ASCII character.

An initial cut at the design resulted in a 12-chip circuit with a timing budget that appeared feasible. A second cut at the design resulted in an operational prototype requiring ten chips including the three conversion PROMs. At the suggestions of W6FFC, a circuit was developed to provide automatic generation of a Baudot LETTERS shift character following a LINE FEED. This feature eases generation of proper end-of-line routine when using an ASCII keyboard and is well worth the three additional chips required. The final design uses 15 chips and was dubbed the ASCII to Baudot Converter – version 1 (ABC-1).

The logic diagram of the ABC-1 is illustrated in Fig. 1. Fig. 2 is the converter timing diagram. Interface with the UT-4 is shown in Fig. 3. Note that one additional 7400 chip (IC13) and one switch (S9) must be added to the UT-4 circuitry to provide ABC-1 interface while maintaining full capability of the UT-4 in the originally intended application.

Features

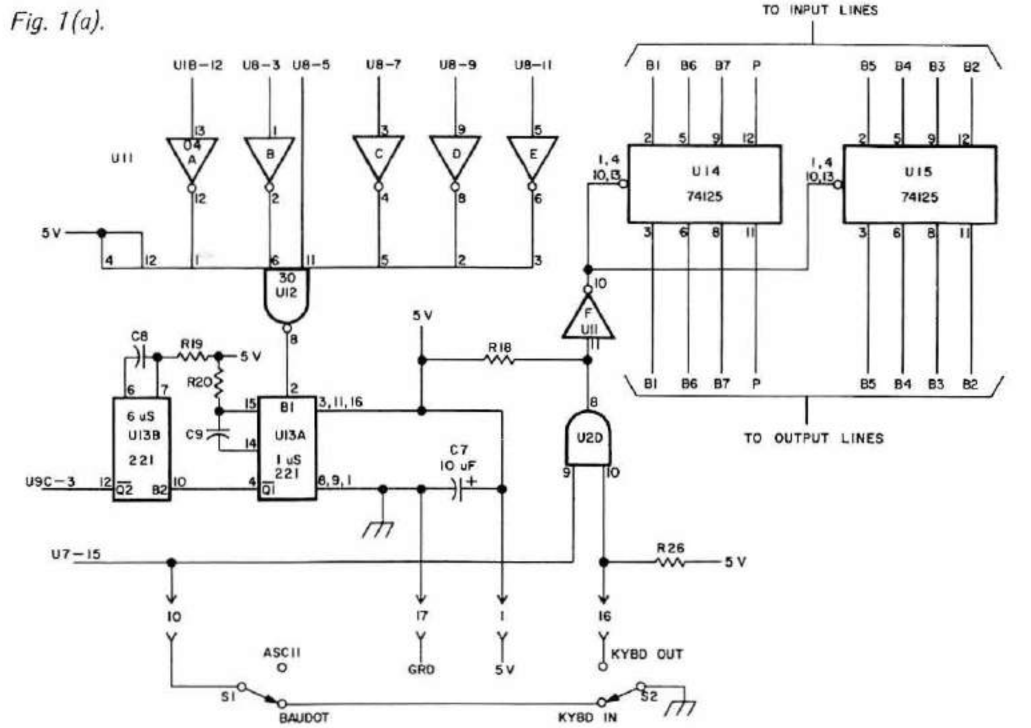
1. Converts all ASCII characters that have Baudot equivalents to the proper Baudot character.

2. Converts all non-equivalent characters to a Baudot BLANK unless otherwise programmed in the appropriate PROM.

3. Converts both upper and lower case ASCII alphabet characters to the equivalent Baudot character.

4. Provides automatic Unshift-on-Space for Baudot machines.

Fig. 1(a).



Figs. 1 and 1(a). U1, 3, 10, 13: 74221. U2: 7409. U4, 5, 6: 8223. U7: 74155. U8: 74365. U9: 7410. U11: 7404. U12: 7430. U14, 15: 74125. R1-5, 7-11, 18, 26: 4.7k. R12: 30k. R13-17, 19, 20: 15k. R21-25: 10k. C1: 330 pF. C2, 3, 9: 100 pF. C4: 210 pF. C5, 6, 8: 560 pF. C7: 10 uF, 10 V. Notes: ASCII inputs B1 through B7 are positive logic (Mark = High level). U4 is alphabet PROM that converts both UC and LC ASCII to Baudot. U5 is control function PROM for Carriage Return, Line Feed, and Bell. U6 PROM converts numerals, punctuation, and space bar. Most ASCII characters with no Baudot equivalent convert to a Baudot BLANK character. U11, 12 and 13 generate a Baudot LTRS shift function immediately following a Baudot LF, thus providing a standard end-of-line routine capability of CR, LF, LTRS. Baudot outputs B1 through B5 are positive logic (Mark = High level). U14 and 15 provide direct ASCII throughput when S1 is in ASCII position. For U1, 3-8, 10 and 13, Vcc is on pin 16 and Gnd is on pin 8. For U2, 9, 12, 14 and 15, Vcc is on pin 14 and Gnd is on pin 7.

5. Provides Automatic Letters Shift after LINE FEED.

6. Provides a Baudot Letters Shift on receipt of ASCII "RUBOUT" or "UNDERSCORE".

7. Provides a Baudot Figures Shift on receipt of ASCII "UP ARROW" or "~".

8. Provides for direct throughput of ASCII code.

9. Provides 3-state buffered data outputs for data bus applications.

10. Easy interface to the UT-4.

Functional Description

Parallel format ASCII data is applied to inputs (address lines) of 8223/74188 PROMs U4, U5 and U6. Note that only bits 1 through 5 are used for addressing the PROMs. Bits 6 and 7 are applied to 2-line to 4-line decoder U7. The binary state of bits 6 and 7 are decoded by U7 to provide an enable signal to pin 15 of the appropriate PROM. Decoding of ASCII bits 6 and 7 is arranged by means of U2B so that both upper and lower case ASCII alphabet will be converted to the equivalent Baudot character.

PROM output data (in Baudot code) bits 1 through 5 from all three chips are "wire-or'd" and applied to 3-state buffer U8. If U8 pins 1 and 15 are both low, data from the selected PROM passes through U8 and appears at the output of the converter.

Simultaneously with the appearance of ASCII data at the inputs of the PROMs, the keyboard strobe signal is applied to U1A. U1A and B provide a total strobe delay of approximately 7 microseconds. At the end of this delay period, the strobe signal appears at the output of U2A. When the ABC-1 is connected to a FIFO such as in the UT-4, the delayed keyboard strobe signal causes a "shift in" signal to be applied to FIFO pin 17. Because the data at the output of ABC-1 chip U8 is already present at the FIFO data inputs, this data is entered into the FIFO as a parallel format Baudot character.

The preceding paragraphs describe what happens in the converter when no case shift is required. Let us say that the character converted in the previous example was the character "R". Let us now assume that the next ASCII character from the keyboard is a period. Conversion of this character to Baudot code requires that it be preceded by a FIGS shift character. The states of bits 6 and 7 in the ASCII period character cause PROM U6 to be selected for punctuation characters (numeral conversion also takes place in this PROM). PROM U6 *output* bits 6 and 7 are Low and High respectively for a period character and are applied to the case shift detector latch U9A,B where pin 12 of U9A was Low for the previous character R. Bit 7 is High and so has no effect on the latch. Bit 6 is Low, causing U9A,B to change state, and pin 12 goes High. This Low to High transition is applied to input B1 of 1-shot U10A, generating a 6-microsecond wide FIGS shift gate at U10A pin 4. This gate performs three functions. It inhibits U8, causing U8 outputs to revert to the 3rd (high impedance) state, and because of the current sources through R21-R25, U8 output bits B1, 2, 4, 5 go High. Bit 3 goes Low because of the inverted (Low) output of U2C which is the second function of the FIGS shift gate. The 3rd function of this gate is to generate a "Case Shift Strobe" pulse by means of U3A,B and U2A. This strobe is delayed 2 microseconds by U3A, permitting the parallel data at U8 output (bits 1, 2, 4, 5 High and bit 3 Low = Baudot FIGS shift) to settle to a static condition before being entered into the FIFO by the strobe signal.

So far, approximately six microseconds have elapsed since the ASCII data and strobe for the ASCII character "period" appeared at the input to the converter. At the end of the 6 microseconds, U8 is enabled, and U2C output returns to a high level. At this time the Baudot character for period (B1, 2 = Low, B3, 4, 5 = High) is present at the output of U8. One microsecond later the delayed (7 microsecond) keyboard strobe from U1B appears at U2A pin 11 and now the Baudot period character is entered into the FIFO. Generation of a LETTERS shift character in U10B is similar to the foregoing except that U2C output remains High (U8 outputs B1 through B5 are all High).

Thus it is apparent that all normally converted characters are delayed by seven microseconds within the converter before being strobed into the FIFO. If a case shift character is required, it is generated and strobed into the FIFO during the seven microsecond delay interval.

U11, 12, 13 and U2D form the "Letters shift after Line Feed" circuit. U11 and U12 detect the presence of a Baudot Line Feed character at the output of U8. The output of U12 goes Low when the normal delayed strobe pulse from U1B appears at U11A. After a one microsecond delay through U13A, U13B generates a six microsecond pulse that is applied through U2D and U9C to start the generation of a Letters shift character as previously described. A non-printing character such as Letters shift, following a line feed, gives the machine time to return to the left margin before printing the next character.

In Fig. 2, waveform 11 of the timing diagram is a composite of the various conditions at the converter output strobe line. The "Typed character strobe" (center pulse in the pulse train) will appear on the strobe line every time a character key is pressed on the keyboard. The case shift strobe and LSAL strobe are shown as dotted lines, indicating they will appear only under certain conditions. Depending on previous conditions, all three strobe pulses can appear in the sequence illustrated when the converter receives an ASCII LINE FEED character.

Construction

The timing components should be kept clear of the trigger inputs on the one-shot multivibrators. Five percent tolerance dipped mica capacitors and five percent ¼ Watt resistors should be used in the one-shot timing circuits. Three ABC-1 converters have

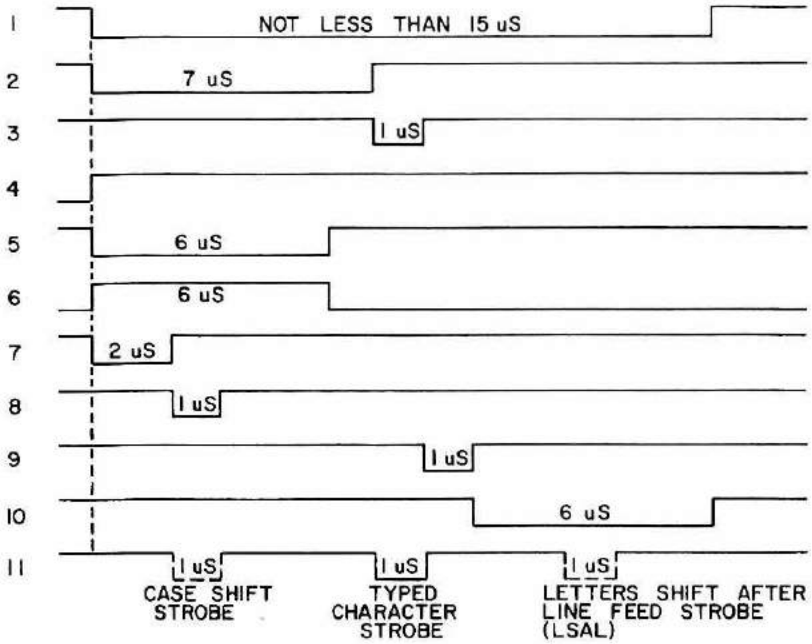


Fig. 2. Timing diagram. 1 – Strobe in, U1A-1. 2 – Strobe delay, U1B-10. 3 – Delayed strobe, U2A-13. 4 – Figs shift, 1-shot U10A-2. 5 – Figs shift gate, U10A-4. 6 – Case shift gate, U10A-4. 7 – Case shift strobe delay, U3A-4. 8 – Case shift strobe, U2A-12. 9 – LSAL gate delay, U13A-4. 10 – LSAL gate, U9C-3. 11 – Strobe output, U2A-11.

been constructed and the timing was well within tolerance using off-the-shelf five percent components.

Use of a printed circuit board makes construction much easier and decreases chances of wiring errors. Even so, the PC board has a high component density and traces are very close together. Great care should be exercised during assembly and soldering to prevent errors in component location, IC orientation, and solder bridges. Sockets or molex pins are recommended for the ICs.

PC Boards and PROMs

EDI[4] has been authorized to make a PC board available for the ABC-1. This is a glass epoxy, double-sided, plated-through hole circuit board. It will fit a standard .156 spring 18-position double-readout edge connector. Boards only, or complete parts kits including the three pre-programmed PROMs, are available.

If you already have a UT-4, Fig. 3 shows the changes required to interface with the ABC-1. If desired, EDI has a modified UT-4 PC board with these changes incorporated. Order "UT-4 IF" PCB/kit. This PCB fits the same type edge connector socket as the ABC-1.

Troubleshooting

Most comments on troubleshooting in the KBI-1 article apply to the ABC-1. The converter is a fairly complex circuit with critical timing parameters. If a scope is not available, it is mandatory that the associated UT-4 be operational in order to check converter operation.

Strobe and data lines from the KBI-1 outputs to the ABC-1 inputs and from the ABC-1 outputs to the UT-4 should be less than 20 inches in length. Transmission line techniques must be used for longer lines as described in the KBI-1 article.

Logic levels at the seven data inputs to the ABC-1 must be stable at the time the keyboard strobe goes Low at U1A-1 and U7-14. As indicated in the timing diagram, the minimum width of the strobe pulse is 15 microseconds. If your keyboard strobe is pulsed with a period, for example, of 5 microseconds, then U14 of the KBI-1 must be used to stretch this 5 microsecond pulse to at least 15 microseconds. Another possible source of improper operation is errors or omissions in the switching and control circuitry (S1, S9) as shown in Fig. 3.

To use the Repeat message function in the UT-4 during operation with the KBI-1/ABC-1, PRELOAD switch S5 is set to PRELOAD, KYBD switch S9 to IN and the message is typed into the UT-4 memory from the keyboard. KYBD switch S9 is then set to OUT and REPEAT switch S8 is set to REPEAT. Then PRELOAD switch S5 is set

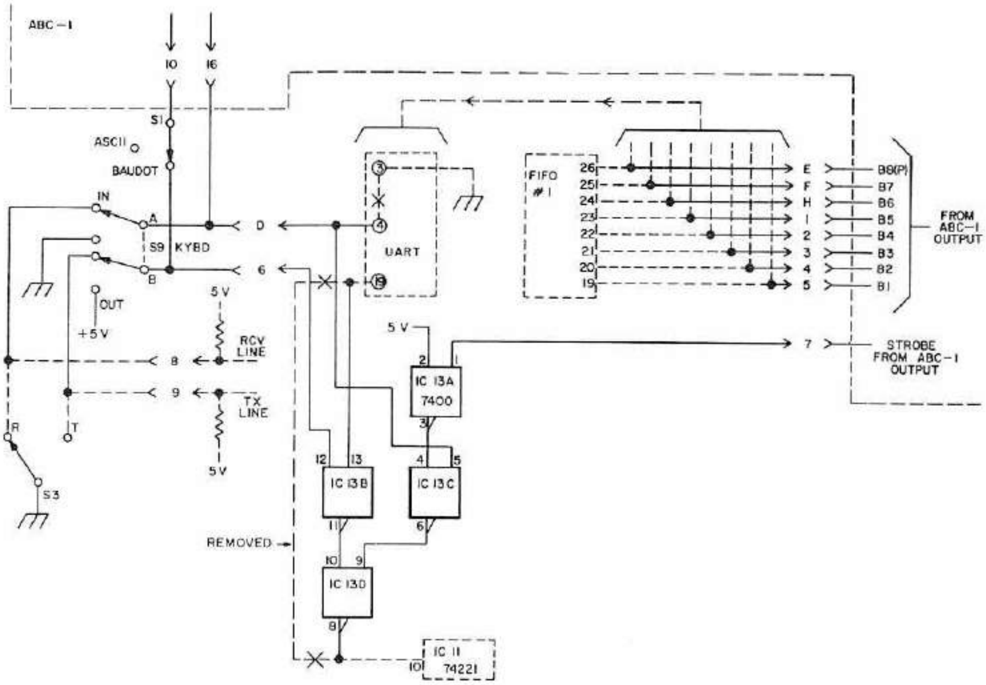


Fig. 3. Changes required to interface UT-4 to ABC-1. Notes: Dotted lines show existing circuits in UT-4. Solid lines indicate added circuitry to accommodate ABC-1. These additions change the UT-4 PCB to UT-4 IF PCB (IF = Interface). ASCII/Baudot switch S1 is the same as S1 in ABC-1 schematic. Keyboard In/Out switch S9 is added to UT-4 control switching. UT-4 IF has all the features of UT-4 except that Space switch S7 is deleted. UT-4 IF edge connector references are for the EDI printed circuit board.

to NORMAL and the message is continuously recirculated through the UT-4. REPEAT switch S8 also permits local copy on your printer during the repeat sequence when using the UT-4 IF circuit board. KYBD switch S9 must be returned to IN if you wish to continue using the keyboard after completing the repeated sequence.

Acknowledgements

In addition to the valuable suggestions contributed by W6FFC and WB6WPX, thanks is due Peter K6SRG, for his many helpful comments and for checking out the first prototype PCB. I am indebted to Steve WA6TVA, for pointing out the need to force the case shift latch to the letters state when auto letters after line feed is generated. Appreciation is also extended to many others who offered suggestions and encouragement, including K2SMN, K3TML, WA5NYY, W6GQC and WA7ARI. ■

References

[1] Ellsworth, C. A., "Using A Bargain Surplus Keyboard" (The KBI-1), *73*, January, 1976.

[2] Lancaster, Donald M., *The TTL Cookbook*, 1974, p. 153.

[3] Hoff, Irvin M., "The Mainline UT-4," *RTTY Journal*, March, 1975, p. 4.

[4] Electronic Development, Inc., PO Box 951, Salem OR 97308.

I/O Editorial
from page 87

which are not needed, as a key to copyright infringement prosecution. We will need games . . . chess, backgammon, Star Trek, and so forth . . . business programs such as inventory, general ledger, accounts payable, accounts receivable, mailing list, and so forth. We will have to have a statement from you that your work is original.

Hopefully this system will provide the money to encourage programmers to work overtime to provide the software we need to sell small computer systems. By bringing a system to this presently chaotic aspect of computers, perhaps we can help the field to grow more rapidly.

ATTENTION, CLUBS!

One of the big problems for many club meetings is getting interesting material . . . so I've been watching out for interviews which I might get on tape which could be of interest to clubs.

The first tape I have available is an interview with Ed Roberts, the president of MITS . . . the first manufacturer of microprocessor kits to hit the jackpot. His Altair 8800 computer kit came out about a year and a half ago, just as MITS was in serious shape as a result of the plummeting prices in the calculator market. Their first year's sales of computers was about ten times what they estimated, and they are just now beginning to get caught up with events.

Ed's story is an interesting one, and it will give you a lot of insight into the manufacturer's side of things . . . it will also familiarize you a lot with the computer scene . . . what microcomputers are being used for and where the market seems to be going. Hear about some exciting technical developments which are coming.

The tape is on cassette only and is one hour long. It was made with top notch professional equipment, so the sound will be as good as your cassette player. Order Cassette Interview I . . . and never mind the usual prices for such tapes, normally around $12.95 . . . *73* is in the magazine business, not in the tape business, so your cost is only $3.95 postpaid.

This tape may also be of interest to repeater groups which run club bulletins and news of interest to members over the repeater.

INDEX